THE CP/M USERS JOURNAL $2.95

Microsystems

September/October 1982

The Problems And Some Successes In The Transporting Of Large Languages To Microcomputers

MORROW DESIGNS

5¼" DMA Hard Disk Systems
M5 $2,195 M10 $2,795 M16 $3,495

Morrow DMA Controllers Supercharge Disk Systems.

What EDN said: "The (Disk Jockey™ DMA floppy-disk controller) offers two to three times the performance of comparable controllers." And that's for either 5¼" or 8" floppy-disk drives operating from an S-100 (IEEE-696) bus.

How we do it? With channel drive that's almost identical to IBM 370® channel controllers. The DJ/DMA uses 24-bit addressing. The host writes commands into memory. The DJ/DMA picks up commands from the host processor via memory on the system bus, and transfers data during DMA cycles. Channel commands may be located anywhere in the 24-bit address range. Upon completion of the command, the controller returns status. It may also generate an interrupt. Chained commands allow the controller to return status, or to execute a number of commands in succession. The controller board also contains logic which allows other IEEE-696 temporary bus masters to contend for memory cycles. That's the Morrow "channel concept." Fast and Simple.

On-board Z-80A.* By managing both memory and disk transfers, the resident Z-80A allows reading or writing to almost any floppy-disk media. 8" or 5¼". Single or double density. Single or double-sided. And up to eight drives per controller board (no more than four of each type).

System compatible. Disk Jockey DMA sub-systems are compatible with all IEEE-696, S-100 systems (such as the Morrow Decision I™). And, with most S-100 like systems.

Faster Winchesters. Hard disks put micros in the mini category. Now, Morrow's channel driven DMA controller concept makes Winchesters lightning fast. Which Winchester? Industry standards—Seagate's 5¼" ST506/512, or 8" drives from Shugart/Quantum.

DMA Transfer (Burst Mode). The Morrow Direct Memory Access Hard Disk Controller (HDC/DMA) picks up commands from the host processor via memory on the system bus. Commands are accessed and data is transferred during DMA cycles. Commands and data transfers may occur anywhere in the 24-bit address range.

Interrupts. The controller can generate an interrupt at the end of each command and/or at the end of each command chain.

Imbedded μP. An on-board 8X300 supervises data transfers between the Winchester drive(s) and main memory. Microcode in this 7 MHz bipolar microprocessor implements the command structure of the controller.

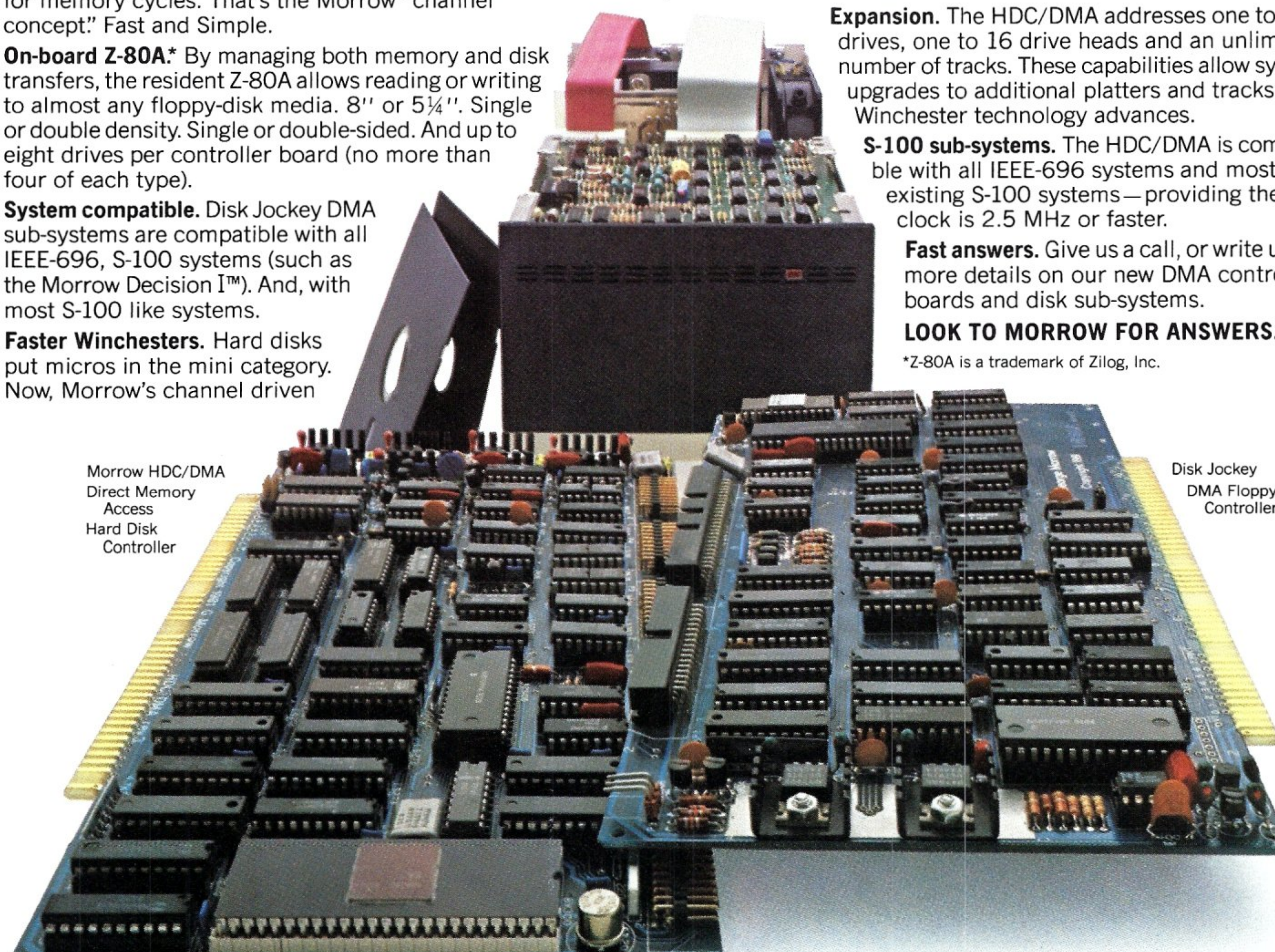
Expansion. The HDC/DMA addresses one to four drives, one to 16 drive heads and an unlimited number of tracks. These capabilities allow system upgrades to additional platters and tracks as Winchester technology advances.

S-100 sub-systems. The HDC/DMA is compatible with all IEEE-696 systems and most existing S-100 systems—providing the bus clock is 2.5 MHz or faster.

Fast answers. Give us a call, or write us for more details on our new DMA controller boards and disk sub-systems.

LOOK TO MORROW FOR ANSWERS.

*Z-80A is a trademark of Zilog, Inc.



MORROW DESIGNS
5221 Central Avenue, Richmond, CA 94804
(415) 524-2101, TWX: 910-382-8132, (Power Key-Richmond)

CIRCLE 20 ON READER SERVICE CARD

FDC-I

SBC-I

SBC-I

SYSTEMASTER®

The Next Logical Step

Teletek FDC-I, The First Step
A CPU, floppy disc controller and I/O all on one board. Until SYSTEMASTER® the most powerful IEEE-S-100 board on the market.

Teletek SBC-I, The Second Step
A board designed for use in high performance, multiprocessing systems. SBC-I combines its own CPU, serial and parallel ports, 128K of segmented memory and FIFO.

SYSTEMASTER®, The Next Logical Step
Teletek's newest product — a breakthrough in S-100 board design. SYSTEMASTER® is a complete system. No other boards are required. It doesn't even require a bus!

But, when used with an S-100 bus, SYSTEMASTER® has even more capabilities. You can readily expand to multi-user or multi-processor systems using SYSTEMASTER® as the master board.

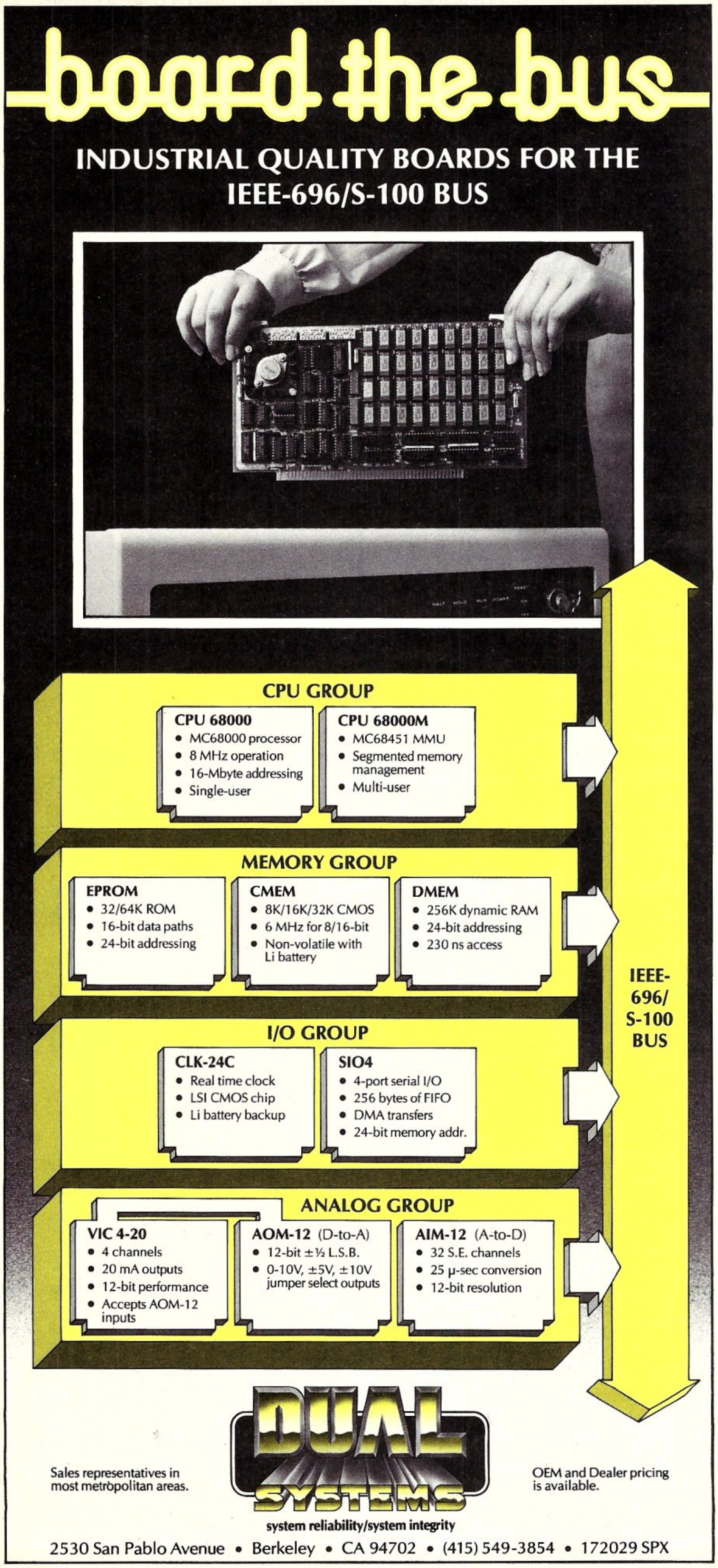
SYSTEMASTER® Features:
- Floppy controller: controls mini and/or maxi drives; variable independently-adjustable pre-write compensation.
- CPU: 4MHz Z80A
- Two RS-232C ports.
- Two full parallel ports.
- 64K bank-selectable memory variable from 32K to 60K. Remaining memory is fixed at top of 64K address space.
- DMA: allows transparent data transfer to floppy disks and other devices without CPU wait states.
- Relocatable EPROM/ROM/PROM: 2K, 4K or 8K with deselect capability.
- Real-time clock.
- Operating systems available: Digital Research or Turbodos.
- Parallel ports may be connected to Teletek's PSC board for RS-422 capability.

TELETEK

9767F Business Park Drive Sacramento, CA 95827 (916) 361•1777

Contents

Volume 3/Number 5 September/October 1982



CIRCLE 6 ON READER SERVICE CARD

Microsystems

MICROSYSTEMS (ISSN #0199-7955) is published bimonthly by Ahl Computing, Inc., a subsidiary of Ziff-Davis Publishing Company, One Park Avenue, New York, N.Y. 10016. David Ahl, President; Elizabeth B. Staples, Vice-President; Selwyn Taubman, Treasurer; Bertram A. Abrams, Secretary.

Second Class postage paid at New York, NY 10016 and at additional mailing offices. POSTMASTER: Send address changes to MICROSYSTEMS, PO Box 1987, Morristown, NJ 07960. Subscriptions are $24.97 for 12 issues. Canadian prices are $5.00 per year additional; other foreign $8.00 per year additional (U.S. currency only).

Editorial correspondence is welcomed and should be sent to: MICROSYSTEMS, One Park Avenue, New York, NY 10016. Phone: (212)725-6856.

For information on commercial advertising, write to: MICROSYSTEMS Advertising Dep't., One Park Avenue, New York, NY 10016, or call Jeff Weiner at (212)725-7957.

MEMBER MITA

MULTI-USER HARD DISK SYSTEM—$10,500.

Think Now!

Multi-user MP/M™ 8-16* runs 8 bit CP/M® software better than it's ever been run before.

- **BIG**—over 61K! That's over 6K more user space than conventional CP/M 2.2™!
- **FAST**—DMA floppy and hard disks with cacne buffering and high speed CPUs.
- **MULTI-USER**—enhanced and fully compatible implementation of MP/M 86™ runs both 8 and 16 bit programs at the same time! 8 bit programs you can run include: WordStar,® dBASE II,™ Whitesmith's C, Microsoft™ BASIC, CB80,™ and PL/I-80.™

Think Ahead!

MP/M 8-16 runs 16 bit CP/M 86™ software, so you can get what's coming!

- Upgrade effortlessly from 8 to 16 bit software (for example, from CBASIC™ to CBASIC 86™) without leaving your programs or data behind. You get complete file compatibility between 8 and 16 bit CP/M. Other systems make you choose between 8 and 16 bits. **MP/M 8-16** systems have two CPUs—one 8 bit, one 16 bit. You get it all. Now. And tomorrow.
- 16 bit programs available now include: CBASIC 86, SuperCalc 86,™ Microsoft FORTRAN 86 (ANSII 77), CIS COBOL 86.™
- Even the hardware is upgradable. Our boards are compatible with the IEEE 696 bus (and we've got 20 bus slots!). Our state-of-the-art systems can keep pace with the state-of-the-art.
- All CompuPro boards in our **MP/M 8-16** systems are covered by a factory backed two year phone replacement warranty.

G&G Engineering is an authorized CompuPro systems center.

Think G&G ENGINEERING!

1922 Republic Avenue, San Leandro, California 94577 (415) 895-0798
351 California Street, Suite 515, San Francisco, California 94104

*MP/M 8-16 is a proprietary implementation of Digital Research's MP/M 86 operating system, configured for CompuPro by G&G Engineering.

CIRCLE 12 ON READER SERVICE CARD

Editor's Page

Changes at Microsystems

MICROSYSTEMS is growing at a strong, consistent rate and we feel that it is improving in content as well as appearance. The reader survey which we conducted in the early part of the year attests to this view. I will report, in detail, in the next issue on what our readers told us. In the meantime I will fill you in on the changes at MICROSYSTEMS.

In the early part of this year MICROSYSTEMS became part of the Ziff-Davis family of magazines. Z-D is one of the largest magazine publishing houses in the world (e.g., *Popular Electronics, Popular Photography, Stereo Review, Psychology Today*, etc., etc.). MICROSYSTEMS is now part of Z-D's rapidly expanding group of information-processing publications—a group that includes *Data Sources, Data Decision, Computer Intelligence, Small Business Computers, Creative Computing*, and *Sync*.

Z-D is investing a considerable sum in MICROSYSTEMS. The goal is to make it "the" leading magazine for the sophisticated microcomputer user. Big investments are being made in promotion to subscribers and advertisers, as well as enhancing content and appearance. The print run for the magazine has been increased (the July/August issue was 22,000 and this issue's run should be significantly larger).

Z-D has also enlarged the MICROSYSTEMS staff. Most notably Z-D has hired a technical editor for MICROSYSTEMS. He is Chris Terry. I am sure you will all recognize Chris's name. He has written many of the articles in past issues of the magazine . . . most notably the very popular series titled "The CP/M Connection," an in-depth tutorial for programmers interfacing to CP/M. Chris has also written numerous software and hardware reviews and a popular article on sorting techniques. He was previously employed by the Systems Research & Development Division of Dun & Bradstreet, where he prepared documentation for systems and application software. Chris is also the co-author of a book on micros recently published by Van Nostrand.

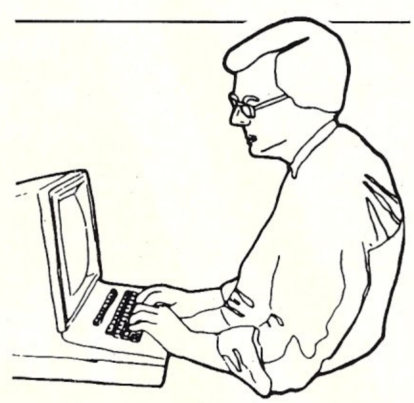
You may have also noticed that, starting with the July/August issue, we introduced a "reader service" card, something readers and advertisers have been pleading for. This is a very expensive undertaking and frankly, until the Z-D acquisition, we were not in the financial position to undertake it. You will also notice an improvement in the appearance and printed quality of the magazine. This is because magazine production has been moved to the Z-D production department at One Park Avenue, New York City and we have switched to a new printer.

If our circulation and advertising increase as expected we will be increasing the size of the magazine and publishing it more frequently. This will mean a significant increase in available editorial content space, allowing us to broaden the scope of MICROSYSTEMS, though the main emphasis will continue to be on support for CP/M users. We will also cover any and all hardware systems that run CP/M, which will include S-100/IEEE-696 systems, single-board systems (e.g. Osborne) and limited bus systems (e.g. IBM-PC). We will also be giving some coverage to other operating systems such as MS-DOS, SB-86, MP/M, Turbodos, Oasis, Xenix and other Unix-like operating systems.

The technical level of the magazine will continue to be aimed, as in the past, at the sophisticated user of microcomputers. It will not cater to the beginner or the game player. After all, there are several other magazines currently serving these users.

Another result of the Z-D acquisition is that we have considerably improved the rates we pay authors. Incidentally, we are looking for more articles, so write or call us if you would like to receive a copy of our author's guide. In addition to the areas listed above, we are looking for articles on networking, hardware and software interfacing, communications, multiprocessing and multiuser systems, software development tutorials, graphics and "hints and kinks," etc., etc.

Happy reading.



Sol Libes

*Without sacrificing your 8-bit investment in Hardware and Software

Our DISCOVERY Multiprocessor is the **first and only** multi-user multiprocessing microcomputer system that allows for the simultaneous operation of **both** 8-bit , Z-80 based and **16-bit, 8086/8087** based processors within the same system. Furthermore, it is now possible to combine individual DISCOVERY Multiprocessor stations into **Local Area Networks** with capability surpassing most minicomputers and mainframes.

THE END USER is guaranteed access to the World's most abundant supply of computer languages and software. In DISCOVERY, the industry standard **CP/M-80 and CP/M-86**** run simultaneously and concurrently while maintaining compatibility between 8-bit and 16-bit databases. With DISCOVERY, the new generation of software designed specifically for multi-user applications is available for your use... **Right Now!**

As **A SYSTEM HOUSE**, you will find that DISCOVERY easily brings all your existing 8-bit development tools and high-level applications software to the strength of the 16-bit World. Just think, access to a million bytes of addressable memory and lightning fast execution speeds. With the powerful file and record locking facilities of our own **dpc/os**©** (Distributed Processing Computer Operating System), you can create flexible solutions to meet your customers' needs. Let your sales grow into the ever expanding multi-user market.

DEALERS and DISTRIBUTORS, you will find it easy to sell our DISCOVERY system to your customers with the assurance that any time they chose, they can grow painlessly into the **Sixteen-Bit** World and into **Networking.** The DISCOVERY is the only system in the microcomputer marketplace that will provide your customers with the natural growth path...from 8-bit single user computers to full-blown multi-user networks of **16**-bit work stations and with up to **SIXTEEN** users in each complex!

**CP/M-80 and CP/M-86 are TMs of DIGITAL RESEARCH INC.
dpc/os© is a TM of ACTION COMPUTER ENTERPRISE, INC.

Take ACTION! Call us NOW... at (213) 793-2440

Systems complete with Hard Disks available for immediate delivery. Dealer, Distributor & OEM inquiries are invited.

Action Computer Enterprise, Inc. ○ 55 West Del Mar Blvd. ○ Pasadena, CA 91105 USA
TWX-910-588-1201 ACTION PSD ○ Telephone (213) 793-2440
On the East Coast: Please Contact MicroSystems International ○ (617) 655-9595

CIRCLE 78 ON READER SERVICE CARD


GREAT IDEAS...
Down to Earth
Products
HARDWARE
SOFTWARE
SIERRA DATA SCIENCES
(AVAILABLE SOON)
SDS-S100 CHASSIS
6 Slot Motherboard
Complete power supply
Utilize regular or Thinline drive
SDS-MULTIPLEXER/DISPLAY
Three RS232C 1 to 2 Switches
Two Seven Segment Status Display
(Can be used as a line monitor for data communications link)
(route RS232 to one of two devices)
NEC-FLOPPY DISK DRIVE
Double Sided
Single Density/Double Density
Up to 2.4 Megabyte
SPECIAL PRICING
SDS-HARD DISK INTERFACE
Micropolis 1220 Series
Interface Adapter
SDS-SLAVE
SDS-ZSIO
SDS-MASTER
SDS-SINGLE BOARD COMPUTERS
Z80A CPU 64K Bank Switch Memory
2 RS232 Channels 4 Timers IEEE 696 Buss Interface
4 Parallel ports
NEC 765 FDC with PLL to all Shugart compatable drives (SDS-Master only)
CIRCLE 94 ON READER SERVICE CARD
SIERRA DATA SCIENCES Fresno CA / Marketing Division, 21162 Lorain Ave., Fairview Park, Ohio 44126 (216) 331-8500 Telex. 980133 WDMR

News & Views

S-100/IEEE-696 Standard News

Mark Garetz (General Manager of the CompuPro Division of Godbout Electronics) took over the Chairman's position of the IEEE S-100 Standard Committee (P696) in March. Under his direction the changes to the standard were finalized and the standard was sent out to committee members to be voted on in May and was approved. The standard was then submitted to the IEEE Microcomputer Standards Committee (MSC) in June and was approved. The standard now must be submitted to the IEEE Computer Standards Board and then to the IEEE Standards Board for final approval. It is expected that the standard will be formally adopted by year end.

We are making every attempt to get permission from the IEEE to reprint the adopted standard in MICROSYSTEMS, as soon as it is adopted. If we are not able to do this we will be sure to let you know how and where you can obtain a copy.

National Introduces 16-Bit Microprocessor

National Semiconductor has started shipping samples of its 16-bit microprocessor chip set called the NS16016. The CompuPro Division of Godbout Electronics is already working on an NS16016 S-100/IEEE-696 CPU card which they expect to announce officially before year-end. Digital Research has also disclosed that it will develop and market a multitasking version of CP/M for the NS16016.

The NS16016 is actually a 32-bit micro with 16-bit I/O. It executes 8080 instruction codes. Its arithmetic logic unit, internal data paths and registers are 32 bits wide. Further, it supports demand-paged virtual memory. Because the 16016 can run 8080 code, the device can run thousands of immediately available programs.

Early next year National expects to introduce a true 32-bit version of the microprocessor to be called the "NS16032". It will have 16-bit I/O, and Digital Research is considering development of an operating system for this device also. Later next year National expects to introduce the 32032, which will have 32-bit I/O, and a CMOS version of the 16032 in 1984. Samples of their memory management, interrupt controller and floating-point math chips are expected before the end of 1982. A terminal management processor, local area network controller and hard disk controller are scheduled for 1983.

National has already signed Fairchild to a second-source agreement which includes the development of several peripheral ICs. Synertek is also expected to second-source the IC.

National feels that the 16016 and 16032 present a clear migration path from 8-bit to 16-bit to 32-bit microprocessors that is not currently available with either the Z8000, 68000, 8086 or Intel iAPX432. Hence, even though National is late in introducing the device, it expects strong acceptance.

Digital Research News

John Rowley has been appointed the Chief Operating Officer of Digital Research. He has undertaken a reorganiza-

News & Views
continued . . .

tion that "will allow Digital Research to more effectively serve its customers and manage its growth." Gary Kildall will continue as President, remaining in charge of all functions and supervising all strategic and operational management. Rowley will report to him and oversee each functional department and division.

DR has disclosed that it has signed agreements with Hitachi to develop CP/M and several languages for a 68000 system that Hitachi will produce, and an agreement with Intel to develop a multiuser, multitasking operating system for the iAPX286. DR has also eliminated runtime library royalties for all DR languages.

DR will also open regional offices throughout the U.S. and Europe; these will include a demonstration center and full sales and support services. The first office will be opened in Boston with four other offices expected before yearend.

DR has also signed an agreement with Graphic Software Systems, Wilsonville OR to jointly develop graphics products that are consistent with the emerging ANSI standards for computer graphics. Initial products will include a library of graphic primitives.

Dual Processors

CompuPro started it with their 8085/8088 dual processor card introduced almost two years ago. Now there are almost a dozen companies with dual processor systems: Zenith (8085/8088), Cromemco (Z80/68000), Radio Shack (Z80/68000), Vector Graphic (Z80/8088), North Star (Z80/8088) and Dynabyte (Z80/8086) to mention just a few.

SORTING

with CP/M?

The answer is CO-SORT.

CO-SORT is the best general purpose sort/merge routine in the business. Here's why!

- CO-SORT runs with your program—no stopping and restarting and no transfer files to build and move.
- CO-SORT allows smaller data sets to be sorted entirely in the available memory—perfect for interactive programs where sorting must be transparent.
- CO-SORT is fast and efficient—uses the best algorithm for all sorting requirements and eliminates the need to stop and restart the user program.
- CO-SORT is accessible from all languages, allows fixed or variable length records, all data types, any number of keys in any direction, dynamic choice of source and destination, and more.
- CO-SORT is the choice of professional programmers who want rapid program development with the fastest possible execution time throughout.

CO-SORT from Information Resources makes sorting as sophisticated as it has to be and as simple as it should be. Priced at $200 for an original user license, the package includes a powerful driver program for demonstration and general utility sorting. AND, clear in-depth documentation is provided.

For information or to order, write

INFORMATION RESOURCES
Box W
Manhasset, N.Y. 11030

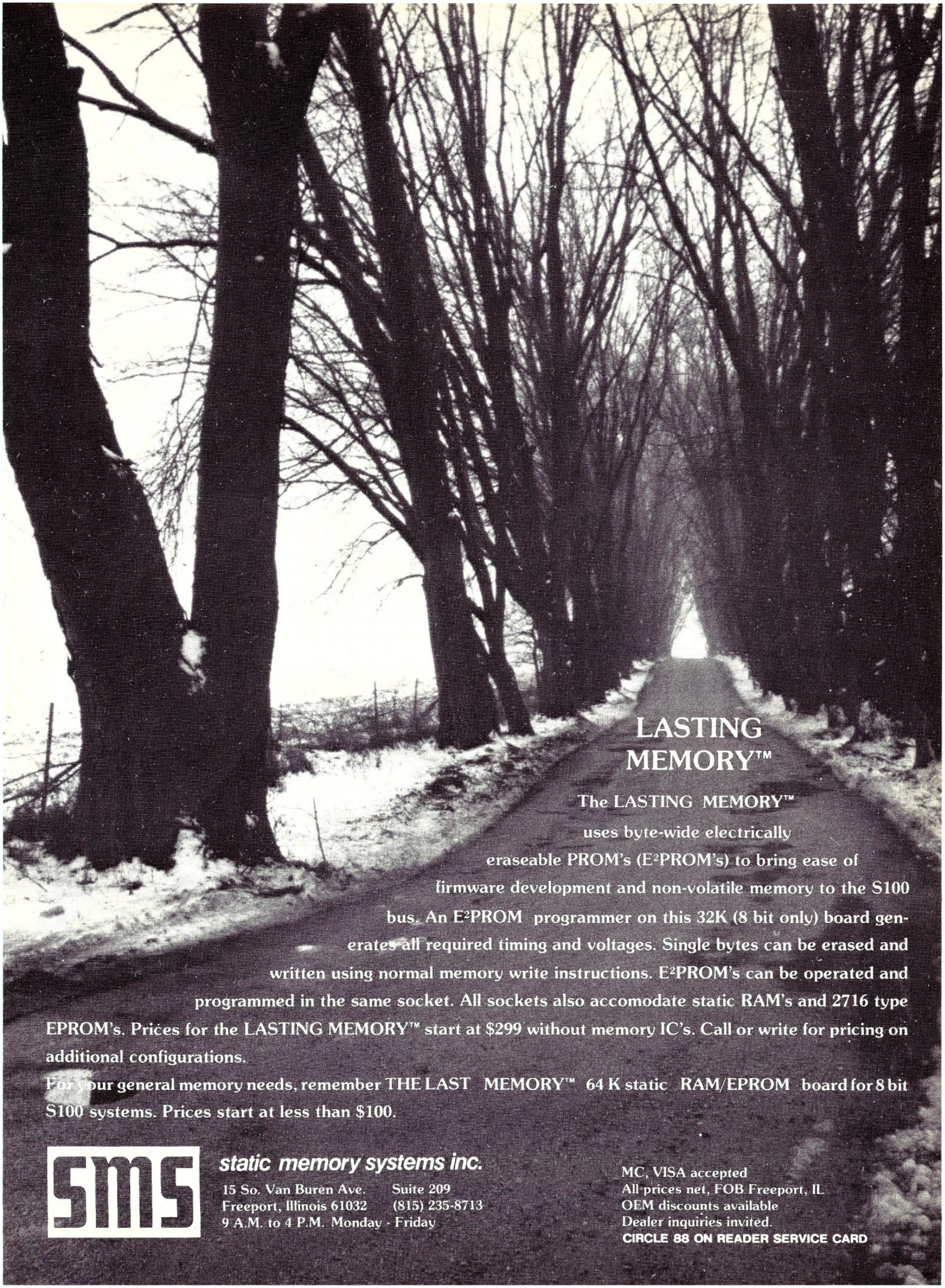
Single-User CP/M Compatible Disk Operating Systems

In the July/August 1981 issue of *Microsystems* (Volume 2, Number 4) I presented a tabular summary of 16-bit operating systems available for 8080 and Z80 based microcomputer systems. The response from readers to that article was very substantial. Hence, it occurred to me to do the same thing for 8-bit operating systems that are compatible with CP/M. After all, how many of you know that there are five other currently available disk/operating systems that are compatible with CP/M?

VENDORS

Digital Research Inc.
Box 579, Pacific Grove CA 93950
408-649-3896

DOS name	*CP/M-80 V2.2*	*I/OS*	*KOS 5.2*	*M/OS-80*	*C/DOS*	*TPM*
Vendor	*Digital Research*	*Info Soft*	*Kontron*	*Mostek*	*Cromemco*	*CDL*
Price	*$150*	*$225*	*Depends on configuration*	*$250 w/ PROMS ¢199 less "*	*$95*	*$79.95*
Released	*August, 1979*	*August, 1977*	*May, 1981*	*May, 1981*	*1977*	*1978*
Romable	*no*	*no*	*no*	*no*	*no*	*no*
Minimum Hardware Required	*20K RAM, console & disk drive*	*24K RAM, console & disk drive*	*64K RAM, 16K video-refresh RAM, video controller& disk drive*	*32K RAM, MDXFLP disk controller & serial or parallel port*	*32K RAM, console & disk drive*	*32K RAM, console & disk drive*
Code used	*8080*	*8080*	*Z-80*	*Z-80*	*Z-80*	*Z-80*
Network Support	*no*	*Asynchronous protocols*	*yes*	*yes*	*no*	*no*
Processor allocation management	*none*	*–system vs. user mode –multitasking; dual tasks –multiuser-structured & task-scheduler –multiprocessing via communications net*	*–multitasking; single foreground task, up to 10 background tasks –no sysgen program furnished*	*–system vs. user mode –multitasking available –sysgen source not furnished*	*none –sysgen source not furnished*	*none*


LASTING MEMORY™
The LASTING MEMORY™ uses byte-wide electrically eraseable PROM's (E²PROM's) to bring ease of firmware development and non-volatile memory to the S100 bus. An E²PROM programmer on this 32K (8 bit only) board generates all required timing and voltages. Single bytes can be erased and written using normal memory write instructions. E²PROM's can be operated and programmed in the same socket. All sockets also accomodate static RAM's and 2716 type EPROM's. Prices for the LASTING MEMORY™ start at $299 without memory IC's. Call or write for pricing on additional configurations.
For your general memory needs, remember THE LAST MEMORY™ 64 K static RAM/EPROM board for 8 bit S100 systems. Prices start at less than $100.
sms
static memory systems inc.
15 So. Van Buren Ave. Suite 209
Freeport, Illinois 61032 (815) 235-8713
9 A.M. to 4 P.M. Monday - Friday
MC, VISA accepted
All prices net, FOB Freeport, IL
OEM discounts available
Dealer inquiries invited.
CIRCLE 88 ON READER SERVICE CARD

DOS name	*CP/M-80 V2.2*	*I/OS*	*KOS 5.2*	*M/OS-80*	*C/DOS*	*TPM*
Vendor	*Digital Research*	*Info Soft*	*Kontron*	*Mostek*	*Cromemco*	*CDL*
Peripheral Management; –Device independent	yes	yes	yes	yes	yes	yes
–DMA	yes	yes	yes	yes	yes	yes
–Spooling	yes	yes	yes	yes	yes	
–Mix storage devices	yes	up to 15 drives, 65Mb each with table-driven handler	yes	yes	yes	yes
–other		uses interrupts & I/O buffering	–uses interrupts –drivers dynamically allocated	–I/O multibuffering –uses interrupts		–any number of drives –table driven
Memory Management –Single contiguous allocation	yes	yes	yes	yes	yes	yes
–Overlays	yes	with program loader	yes	yes	yes	yes
–Chaining	yes	yes binding during link time	yes binding during link time	no segmentation	no	yes swapping

VENDORS, *continued. . . .*

InfoSoft Systems Inc.
25 Sylvan Rd South
Westport CT 06880
203-226-8937

Kontron Electronics Inc.
630 Price Ave
Redwood City CA 94063
415-361-1012

Mostek Corp.
1215 W Crosby Rd
Carrollton TX 75006
214-323-6000

Cromemco Inc.
280 Bernardo Ave
Mt View CA 94043
415-964-7400

Computer Design Labs
342 Columbus Ave
Trenton NJ 08629
609-599-2146

JANUS

Question: When can I use ADA[*]?

Answer: NOW — with JANUS.

The language that is based on the past but looks to the uses of the future.

Ada is available now for your micro-computer.

JANUS is a subset of **Ada** which includes those features sorely missed by programmers on micros. Here is a list to help you decide for yourself.

+ Modular Separate Compilation
+ Single and Double Precision Floating Point Numbers
+ Binary Coded Decimal (BCD) Fixed Point Numbers
+ Integer and Long Integer Numbers
+ Complete String Handling
+ Sequential and Random Access I/O
+ Full Dynamic Allocation and Deallocation
+ Friendly Error Handling
+ An Assembler for interfacing assembly routines
+ A Linker for combining modules
+ True native code is produced
+ ROMable, reenterant code
+ Run-time library source code
+ Low Cost. **JANUS** is more cost effective than any other comparable Ada package
+ Inexpensive Updates
+ No royalties for programs written in **JANUS**
+ No hassle customer service

In short, all pluses. **JANUS** contains everything you need to do fast, structured program development in a micro environment.

JANUS is available for the CP/M, CP/M-86, and MS-DOS operating systems.

Now you too can take a step forward into the future on these computers:

8080/Z80 based systems: (All CP/M) Apple Softcard, North Star, Cromemco, Superbrain, TRS-80 Model II, and all CP/M 8" disk systems.

8086 based systems: IBM Personal Computer, Victor 9000, Seattle Computer System II, Tecmar, Lomas Data Products, and all CP/M 8" disk systems.

8080 or Z80, CP/M (requires 56K memory) — $300.00

8086/8088, CP/M-86 or MS-DOS (requires 96K memory) — $400.00

RR SOFTWARE

specialists in state of the art programming

P.O. BOX 1512 MADISON, WISCONSIN 53701

(608) 244-6436

CIRCLE 11 ON READER SERVICE CARD

DOS name	*CP/M-80 V2.2*	*I/OS*	*KOS 5.2*	*M/OS-80*	*C/DOS*	*TPM*
Vendor	*Digital Research*	*Info Soft*	*Kontron*	*Mostek*	*Cromemco*	*CDL*
File Management						
–named files	yes	yes	yes	yes	yes	yes
–seq. & random organization yes	yes	random only	sequentially allocated clusters	yes	random only	
–allocation type extents	extents	linked list of sectors	extents	ISAM	extents	
–access files from high-level language	yes	yes	yes	yes	yes	yes
–other		ISAM module available		password/security protection		

CompuPro News

The CompuPro Division of Godbout Electronics is planning to introduce several new products in addition to the new NS16016 CPU card mentioned above. First are three complete systems that will include MP/M-II 8/16, SuperCalc and DBase-II software packages. The RAM-22, a 256K static RAM memory card, is also due before year-end.

Morrow & Cromemco Introduce Low-Cost CP/M Systems

Morrow Designs has introduced a CP/M-based system having 64 K of RAM and one 5.25″ floppy (200Kbytes) for $1195 (a second floppy or hard disk may also be installed in the unit). Bundled into the system is $2000 worth of software that includes CP/M, WordStar, Mailmerge, SpellStar, CalcStar, Wedge, and MBasic. Just add a terminal and printer and you are in business. The system uses a single board CPU and is being made in Japan.

Cromemco has introduced a system for $1785 that includes CP/M, word processing, spread sheet and Basic software. The system includes 64K of RAM, 24K of ROM, a 4-MHz Z80, 12″ display, detached keyboard and one 5.25″ disk drive (390Kbytes).

Zenith Introduces S-100 System

Giving up on its own H-8 bus, Zenith has decided to go with the S-100 bus in its new Z-100 system to be introduced this month. The unit contains a single board that contains dual processors (8085/8088), 128Kbytes of dynamic RAM with parity checking, color display (24×80 characters or 500×640 graphics; 50×80 characters available as an option) and disk controllers, three I/O ports and five S-100 slots. The unit comes in a desk-top cabinet with either a mono-

chrome or color CRT. Also included are two 5.25″ disk drives (floppy or hard disk). MS-DOS will be included (CP/M-80 and -86 are optional).

S-100 Ethernet Controller Introduced

Perex Inc., 1798 Technology Drive, San Jose CA 95110, has introduced the first Ethernet controller for the S-100/IEEE-696 bus. It is a 2-board set called "Filtabyte 2.0" and conforms to the Ethernet specification. It will enable a user to transfer data at 10 Megabits/sec on an Ethernet local area network.

The controller looks at packets of data "on the fly" and, without interrupting the host CPU, determines the action to be taken and takes it. It allows true concurrent processing. The cost is $1295 in OEM quantities.

NEWS BITS

Lifeboat Associates has raised $1 million in venture capital from the firms of Bessemer Venture Partners, Oak Investment Partners and A. David Silver & Co. . . . Vector Graphics Inc., in July discontinued its contract to supply systems to ComputerLand Corp. There are reports that sales slumped when the IBM-PC system was picked up by ComputerLand. V-G will continue to sell systems directly to individual ComputerLand stores. Last year V-G estimated that 9% of their sales was made through ComputerLand stores . . . Tech/Ed Services, 139 Main St., Cambridge MA 02142 plans to publish a directory of S-100 manufacturers and their products. . . . North Star has introduced an 8088 CPU plug-in card for its Advantage computer.

Public Domain Software Libraries News

The SIG/M group has released nine more volumes of public domain CP/M based software bringing their total number of volumes to 64. The new volumes contain the following:

Vol-56 Musicraft System & Documentation
Vol-57 Musicraft Selections
Vol-58 Musicraft Source
Vol-59 PISTOL Language
Vol-60 CP/M Utilities
Vol-61 Pascal-Z Programs
Vol-62 Pascal-Z Programs
Vol-63 Pascal-Z Programs

The SIG/M catalog, listing the contents of all 64 volumes, is $1.50 (domestic) and $2 (foreign) and is available from: SIG/M, Box 97, Iselin NJ 08830.

CPMUG has announced six more volumes, bringing their total to 81 volumes. The new volumes contain:

Vol-76 Re-release of SIG/M Vol-24
Vol-77 Re-release of SIG/M Vol-25
Vol-78 CP/M Utilities
Vol-79 Modem Programs
Vol-80 Cromemco Basic Programs
Vol-81 CP/M Utilities, Editor & Text Processor

The CPMUG catalog is $6 (domestic) and $11 (foreign) and is available from CPMUG, 1651 Third Ave., NY NY 10028.

Replace Your CCP with ZCPR

If you have a Z80 CP/M system and want to improve its performance at next to no cost, then read on. A public domain replacement for the CCP (Console Command Processor) part

of CP/M has been written by a group of hobbyists (Richard Conn, Ron Fowler, Keith Peterson and Frank Wancho) and is called ZCPR. It is available free via many RCPM systems (see listing in *Microsystems* March/April 1982) or from the SIG/M group as Volume 54 ($4). Or you can get it from: MRZ Data Systems Inc., Box 2571, Warminster PA 18974, for $23 (disk + printout of doc file). MRZ for another $10 will create a custom-installed version for you.

ZCPR adds enhancements to the TYPE, DIR, ERA and SAVE functions. New added functions include LIST (output sent to list device), GET (load file at specified address), JUMP and GO. ZCPR includes an excellent manual (doc file) describing installation and use.

User Group News

I have been informed of the existence of the following two new user groups:

Dynabyte User Group: c/o Kelly Borsum, Random Factors Ltd., Box 2875, Durango CO 81301; (303) 247-9306. They publish a newsletter. There is no word on dues.

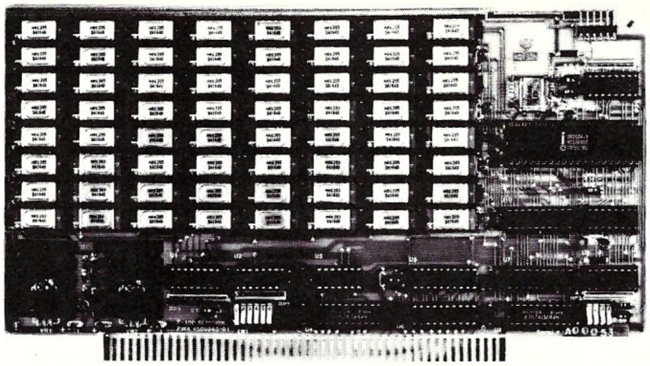
PLUG–PL/I-80 User's Group: c/o Gerry McConnell, Monterey CA 93940; (408) 646-1147. This group plans to distribute PL/I-80 software. No other word on dues or services.

Corrections

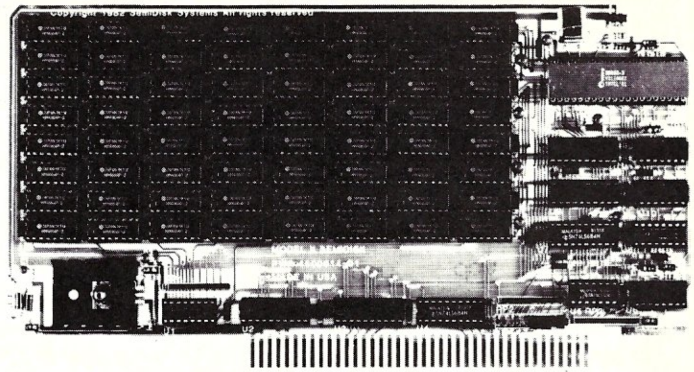
Some errors crept into the diagrams of Chris Terry's article "A Timestamp for CP/M" (Volume 3, No. 2, March/April). We apologize for any inconvenience caused.

1) Figure 3, page 62: the pin numbers of the address lines are reversed. The correct pin numbers are: A3, pin 31; A4, pin 30; A6, pin 82; A7 pin 83.
2) Figure 4, page 64: address line A2 should go to pin 3 of ICs U2 and U3 (NOT to pin 1 as shown). Address line A0 should go to pin 1 of ICs U2 and U3 (NOT to pin 3 as shown).
3) Figure 5, page 64: Pin numbers were omitted from the 74LS367. The inputs from the clock chip should be numbered 2, 4, 6, and 10, starting from the top.
4) Since the 74LS367 can drive only one TTL load, and older systems such as the Altair and IMSAI with front panel may impose more than one load on the DI bus, it may be advisable to use a 74367 (which can drive 10 TTL loads) instead of the LS version.

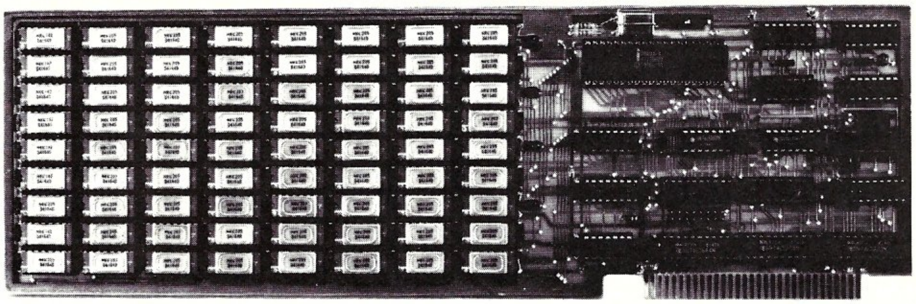
A FULL LINE OF SEMIDISKS



S-100



TRS 80 Model 2



IBM Personal Computer

Do you use your computer? Or does your computer "use" you? Face it, if you're using floppies, your time is being wasted. Because a floppy is an inefficient random access storage device. Each time the processor wants to transfer data, it has to wait an eternity for the disk to rotate and the head to move.

So what do you do? Get a SemiDisk, quick. It's a large capacity semiconductor memory board that is driven by software to operate like a disk drive. Without all the waiting. Do everything you'd do on a floppy or hard disk, with no modifications to your software or hardware. Two board sizes are available: 512K and 1 Megabyte. (the highest density microcomputer memory board in the world) And you can put up to 8 megabytes in a system by adding more storage boards.

What do you need to use it? Just an S-100 system with CP/M 2.2. Or a TRS-80 Model 2 system with CP/M 2.2. Or an IBM Personal Computer. That's it. No special processors, DMA, I/O, or disk controllers are required. Plug it in and run the installation program, and you're on your way. Fast! Even better, we supply full source code to the driver software, in case you'd like to do your own interfacing.

Best of all, the SemiDisk's price won't warp your wallet. Compare specs, cost/megabyte, storage capacity, and compatibility with the competition. You'll see that the SemiDisk is a disk emulator truly worthy of the name. SemiDisk has battery-backup capability, too.

Consider our limited warranty: A full year, covering all parts and labor. Consider our liberal 15 day return policy. Price? $1995 for 512K byte SemiDisk, $2995 for 1 Megabyte SemiDisk. Both from stock. $10.00 for manual. VISA, Mastercard, COD orders accepted. Dealer and OEM inquiries welcomed. (Specify system type and disk format when ordering.)

Someday, you'll get a SemiDisk.
Until then, you'll just have to . . . . wait.

SemiDisk SYSTEMS

P.O. Box GG
Beaverton, OR 97075

(503)-642-3100

NO WAITING

T.M.

Call (503)-646-5510 for CBBS®/NW, a Semi-Disk-equipped computer bulletin board.
SemiDisk trademark of SemiDisk Systems; TRS-80 trademark of Radio Shack

CIRCLE 74 ON READER SERVICE CARD

Letters to the Editor

Dear Editor,

Faced with the need to learn something of 'C', I applied to the local software guru who, amongst other qualifications, subscribes to *Microsystems*. He laid on vol. 2, #6, which purports to introduce readers to C. Reading back to front in the traditional manner, one early finds a full-page advertisement extolling the alleged virtues of *Microsystems*. What is the purpose of such a space waste? How is it reconciled with the claim on page 8 that space is at a premium? The reader of that ad holds in her/his hands a copy of the magazine—surely s/he will soon know, by perusal, more about the possible usefulness of the publication than can be learned from any ad.

If the talents of the *Microsystems* staff were redirected from blurb-writing to editing, we might not find the howler, "Unlike Basic, C is a compiler, not an interpreter" that appears in David A. Gewirtz's article (page 20). Mr. Gewirtz must in some sense believe this nonsense for he repeats it on page 26, even though by page 32 he speaks of a C interpreter and a Basic compiler, thereby affirming that all 4 combinations of language and implementation are not only possible but extant.

This interpreter/compiler confusion forewarns us that Mr. Gewirtz is not going to be up to the task of untangling the obscurantist jargon generated by the jokers at Bell Labs. One can hardly fault an ordinary mortal for being unequal to that task, but then ordinary mortals should not undertake an explication of C.

As an example, consider the simple concept of an "expression." Without excessive rigor, we all know what it means—a bunch of concatenated variables, constants and operators which can be "evaluated" (reduced) to a single quantity, be it numeric, string or boolean. Well, that's not what "expression" means in Murray Hill; an expression can be a statement, and to "evaluate an expression" means to execute a statement, at least sometimes. A clear instance of this debasement of computerese is in "The C Programming Language," D.M. Ritchie et al., *BSTJ*, vol. 57, #6, where we read: "Since assignments are expressions, there is no need for a special assignment statement."

As far as I can see, "expression" is nowhere defined by Mr. Gewirtz, yet it is vital to elementary understanding that this be done. His English translation of the *for-statement* example can inform one that "i=n" is an assignment, not a boolean expression; without the extended notion of "expression' this can only lead to furrowed brows. Of course, brows will generally be in that condition anyway because he blew the example. He should have written:

for (i=n; i!=j/3; ++i)

in order that his translation might assume the value "true."

Perhaps the less said about *switch* the better, since clearly a large glob of text has simply disappeared. About all we can learn from *switch* is that *Microsystems* is not proofread.

A quick tour of the Gewirtz list of operators is similarly depressing. As I do not have the Kernighan & Ritchie book at hand, I hereafter rely on the *BSTJ* piece cited for "official" C information.

*p is *not* a pointer; it is the contents of the location pointed to by p (not nnn, whatever nnn may mean).

+x How can +x "state" that x is positive when in general x may in fact be negative? Could it be an absolute value operator? (Ritchie et al. are silent on + and − as unary operators.)

++, −− contradict the introductory claim that unary operators do not affect stored values. What is one in need of an "introduction" to make of an incrementing or decrementing operator whose action is defined as resulting in the operand remaining *unchanged*? I have joined the ranks of the relative elite by learning that:
x=++i *results in:* x = i = (old i+1)
y=i++ *results in:*
y = old i; i = (old i+1)
(and −− produces symmetrical results), but how can one learn this from Gewirtz?

Turn the page and typographical chaos strikes with a vengeance. It seems that the Gewirtz Printer simply leaves blanks for such symbols as ~, ↑, and |, producing instant garbage for 7 of the operators. Our knowledge that *Microsystems* is not proofread is now expanded: source manuscripts are not read either.

The shift operators are not much use without more information, which may well be hardware-dependent. Do the shifts wrap around (i.e. are they "rotates"?) and do they

include such possible extras as sign, carry, extension or overflow bits?

The examples chosen to illustrate && and | seem particularly inept:
if (a==b && b==c) a=c;
yields the same result as: *if (a==b & b==c) a=c;*
Aside from the (unmentioned) usefulness of strict left-to-right processing, the uniqueness of && (and |) only rears its head when the non-zero-ness occurs elsewhere than bit 1 (or bit 0 if so numbered), and this cannot happen with the boolean expressions chosen.

Presumably, a symbol is missing after *x,y*. As Ritchie et al. are again silent, it might be particularly useful to have some idea of what this operator could be used for. The apparent uselessness may arise from a still-too-restricted notion of "expression."

The definition of *extern* came a cropper somewhere, which is too bad because the official one is fairly murky. It implies that a specific machine storage location will be shared by any number of separate programs which happen to use the same variable name; magic indeed. The berserk word processor (or typesetter) struck again at *pointer to,* which is as good a place as any to lay aside Mr. Gewirtz's errors and turn to his omissions.

He makes no mention of *structures, unions,* or their • operator; of *main* or *default;* or the → operator for pointers. No information is given on operator precedence. He says nothing of the format of function declarations, or the peculiar requirement for a *return* statement within the body, or the awkward business of getting a non-integer value returned or the requirement that all functions be global to the entire program. Missing also is any reference to the "macro preprocessor." Taken at face value, Ritchie et al. (page 2010) appear to claim that assignment of a constant value to a variable name is a C no-no. Rejecting this notion as too preposterous even for the Bell gang, the preprocessor's *#define* seems to reduce to an involved mechanism to squander storage by placing one constant in many locations. Surely this must mean something beyond the grasp of a novice such as myself, but it is apparently too much to hope for explication in what claims to be an introduction.

In case it is useful in placing this criticism in context, my own experience is mostly with low-level hardware, but at one time or another I have worked (or suffered) with Basic, Fortran, Algol-60 (my standard of comparison for high-level languages) and assembler GE-645 through PDP-8 to RCA 1801. The review section of Mr. Gewirtz's article does introduce (as accomplished fact) some of the matters omitted in the "introduction," but this section too is so shot full of error that it seems not worth pursuing.

T.R. Jackson
Box 1
Corinth
Vermont
05039

Don Libes replies:
Thanks very much for your letter and critical interest in the quality of Microsystems.

C is not an easy language to learn. It isn't the kind of language where you can pick up

Move up to Microsystems

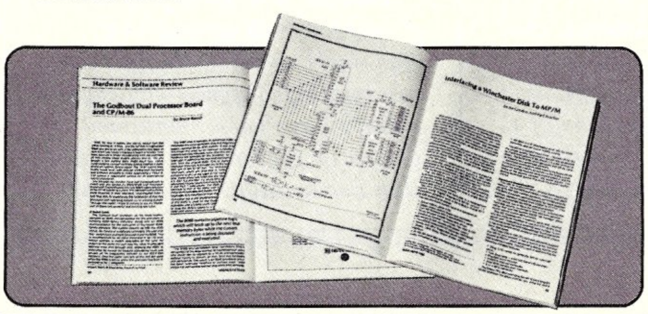
THE JOURNAL FOR SOPHISTICATED MICROCOMPUTER USERS.

Microsystems is not, in any sense, for beginners. Unlike "personal computer" magazines, *Microsystems* is written *exclusively* for expert users and designers in the software and support hardware field.

If you need a reliable, high-level, up-to-date journal devoted to CP/M,® MP/M,® MS-DOS,™ UNIX,® XENIX,™ OASIS and other operating systems, then *Microsystems* is for you. We feature tutorials on the modification and special use of systems, offer short-cut routines, show you debugging procedures, techniques for interfacing, and much more. Our one purpose is to keep the advanced microcomputer professional at the leading edge of DOS development and use.

In a few recent issues of Microsystems you would have seen:

- Reviews of four C compilers: BDS C, Small C, tiny c TWO™ and Whitesmiths C.
- A programming style comparison: Digital Research PL/I-80™ vs. Microsoft BASIC.
- Twenty-seven 16-Bit DOS compared.
- Virtual segment procedures under UCSD Pascal.
- An in-depth look at MP/M-80 II.™
- A simple 6-byte hexadecimal ASCII conversion routine.
- Using CP/M's undocumented "Autoload" feature.
- An explanation of double density disk controllers.
- A comparison of five popular S-100 disk controller cards.
- OS-1—a UNIX-like DOS with a CP/M adapter.

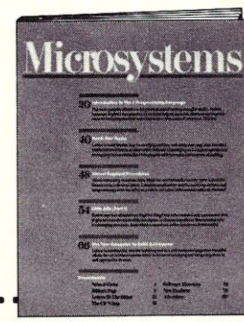


Every issue of *Microsystems* keeps you at the forefront of the industry in customizing software and adapting hardware to meet new requirements and standards. You'll also find industry news, book and new product reviews, plus our regular *Software Directory* listing newly available programs.

To get the high-level systems information you want, join the club of serious microcomputer professionals who subscribe to *Microsystems.* Use the coupon below to order, and save up to 33%!

PROFESSIONALS PLEASE NOTE:

Your subscription to Microsystems may be tax deductible. Check with your accountant.



Trademarks:
XENIX: Microsoft; tiny c TWO: tiny c assoc.; PL/I-80, MS-DOS, MP/M-80 II: Digital Research.
Registered Trademarks:
CP/M, MP/M: Digital Research; UNIX: Bell Laboratories.

Microsystems CN 1987
Morristown, New Jersey 07960

YES, enter my subscription to *Microsystems* for:

☐ 12 issues at $19.97—I save 20%.

☐ 24 issues at $36.97—I save 26%.

☐ 36 issues at $49.97—I save 33%.

Savings based on full 12-issue subscription price of $24.97.

Mr.
Ms. ______________________ 8H05
(please print full name)

Company ______________________

Address ______________________

City ______________________

State ____________ Zip ____________

CHECK ONE: ☐ Payment enclosed. ☐ Bill me later.

Offer valid in U.S. and possessions only. Please allow 30 to 60 days for delivery of first issue. Satisfaction guaranteed or a prompt refund for unmailed issues.

NEW SUBSCRIBERS ONLY

the manual, see the differences it has from, say Pascal, and do it. As Dave Gewirtz amusingly says, it is a high-level low-level language often used for systems work. It requires some finesse to use it well; it requires experience.

You can't learn it from an article in a magazine. In fact, you can't learn it from Ritchie's book, either, in my opinion. All you can hope for is to learn about it. To that end, I think Dave's article accomplishes its task.

Let me reply to your specific points now:

para 2: compiler vs. interpreter. You are right. Dave has made an overgeneralization in trying to be terse. Let us say that while it is by far more typical (and easier) to build Basic interpreters and C compilers, both have been implemented in the opposite fashion.

para 4: As you say, there is an intuitive idea of what an expression is. Hence the author does not define "expression." I don't know why you don't admit a statement such as "a=1" or "fn()" as an expression. These are valid expressions in almost all high-level languages, including many versions of Basic.

para 5: i==j/3. You're right. The example is wrong.

para 6: switch. I don't know what happened to the example code! Somebody goofed during pasteup time.

Page 2 para 1: The BSTJ volume is out of date now and should not be considered "official" in any sense. Indeed, that document states that C is a compiler! My point is simply that new implementations of C are free to differ in functionality from the original UNIX C compiler.

para 2: Your complaints on operators are groundless. His terminology, while informal, is correct. He completely notes the differences in ++ and −−. You might feel enlightened if you were to think about them for more than 10 seconds, but they are really not of any consequence and just serve to confuse people interested in learning about the language. I'm not sure of the unary +. It may be present in one of the implementations he reviewed and therefore he included it for completeness. I'll check with him.

para 3: ~, ↑ and |. You're right.

para 4: shift. "Shift" is not synonymous with "rotate." Status bits are always going to be implementation-dependent, of course!

para 5: & vs &&. As with paragraph 2, you are nitpicking. I think his example is fine. Elaborating further might confuse more people.

para 6: ,. The comma operator exists in the UNIX C compiler and is as the author described.

para 7: extern. The word "is" is missing from between "identifier" and "outside." While not excusable, it's still readable. None of C is "magic" by any means.

para 8: pointer to. You're right. This should have had an asterisk there with the "pointer to" in parens.

Page 3.

Indeed, it is hard for the proofreader (if indeed there was one!) to catch things that are missing. I agree, structures should have been mentioned. Operator precedence and function declarations, on the other

Letters *continued . . .*

hand, are best left unexplained. Operator precedence is not sufficiently different from any other language to warrant explanation. Function declarations are confusing to everybody. I don't know what "peculiar requirement" the return statement is that you mention.

The author should have mentioned the "flat" nature of C, its scoping rules, and its ability to perform recursion. It's unfortunate that this was lacking.

The "macro preprocessor" that old C fans are familiar with was best left out of the article. The facility of saving limited storage space is only one asset that a good preprocessor can add to a programmer's bag of tricks. Any good programming environment will supply one with a preprocessor that handles arguments, condition testing, arithmetic capabilities, string functions, file manipulation AND has nothing intrinsic to do with the language. For example, the Pascal definition speaks nothing of a preprocessor. What one would like to do of course, is to be able to use one powerful preprocessor for both Pascal and C. In fact, a good preprocessor can be used for many applications besides writing programs.

In summary, I think the article was still quite worthy. Granted, there were an unusual number of typographical errors, but the technical content was all correct and well-rounded. Overall, it gave a good picture of what C was like and certainly cleared away any ideas of it being "magic." As for the rest of the article, I thought the reviews were quite enlightening. If I was interested in C and had a question

Letters *continued . . .*

over which one to buy, Dave Gewirtz's article was perfect for laying the answer completely out in front of me.
Better proofreading to come.
Editor

Dear Editor,

I have just received the March/April 1982 issue in which you reproduced a letter I sent to you last year in response to your own call for letters from readers with questions. I understood your own message to suggest that you would try to answer them, rather than simply print them, hence the letter was not only informal, but may be inadvertently misinterpreted and thus slight a fine manufacturer. I mentioned I own a Dynabyte and asked for further information on disc drives and disc errors. However, the errors I experienced I believe were mainly due to careless handling of discs, and after getting proper storage equipment, I have not had one disk error since then—which is more than a year now. The Dynabyte has been highly reliable. I suspect—based on my knowledge of the experience of others—that it is the most reliable of all the S-100 brands I have heard about locally. Furthermore, my own experience and that of colleagues indicates superb, highly conscientious backup from Dynabyte, above and beyond the stated warranty period.

Although my first letter was not intended for publication, I think it is only fair that this one be printed in order to correct any misimpressions it may have created.

Ben Singer
5 Lindbrook Ct.
London, Canada N5X 2L4

Dear Editor,

The SUBMIT replacement by David Cortesi in the May/June 82 issue of *Microsystems* was very interesting for its description of how SUBMIT does its job, and provides a fine base for future improvements. There is a simple patch, however, that can fix one of the defects of SUBMIT with much less effort.

I found the program's urge to translate everything to uppercase the most bothersome problem. Exploration of the object code with DDT revealed a fix. At 036FH of SUBMIT.COM (the version distributed with CP/M 2.2) there is an ANI 5F instruction. Changing the two bytes E6 5F to 00 00 (NOPs) results in a SUBMIT that no longer does case conversion.

Albert S. Woodhull
School of Natural Science
Hampshire College
Amherst MA 01002

Dear Editor,

SUBMIT certainly simplifies compilation and the associated stroke entry for requesting it. However, Mr. Schwab in his article in the May/June issue did not perform the ultimate in simplification—and that surprised me! I use Ward Christiansen's catalog program (CP/M user group disk 40) to keep track of my disks and all I have to say to update the catalog for a disk in drive B is "s s". This is because I have renamed SUBMIT.COM to S.COM and the submit file I refer to is called S.SUB.

Ivan Flores
108 Eighth Ave
Brooklyn NY 11215

Dear Editor,

I think you have a very good magazine; the CP/M + S-100 slot is an important one in the magazine field. Stick with it. I list a couple of areas I would like to see more articles:

(a) More reviews on operating systems (MuDOS or MULTI/OS).

(b) More education-oriented review (what S-100 systems are appropriate for a networked instructional system?).

(c) More construction articles (e.g. a parallel I/O board for adding a printer).

(d) Review on an ST or Shugart 5¼-inch hard disk, and how well it works with a standard controller and software (e.g., CCS).

(e) Articles and programs in Forth.

Dr. David L. DuPuy
St. Mary's University
Halifax, Nova Scotia
Canada B3H 3C3

Dear Editor,

I refer to the evaluation on Data Star by Glenn A. Hart in the May/June issue.

I agree completely that "MicroPro should add a switch which would disable generation of the index file."

One method of getting around this problem and stopping entry "bogging down" is to create an extra field of one character as the "key field" and giving that field the attribute of only being one particular character. In this way it is not time-consuming for Data Star to keep the index file and input is not slowed down.

Robert T. Armstrong
P.O. Box 2623
Toronto, Australia 2283

MEET OUR FAMILY

The InfoSoft Family of MicroComputer Operating Systems

Over 100,000 users can't be wrong

INTRODUCE ME! I AM A:

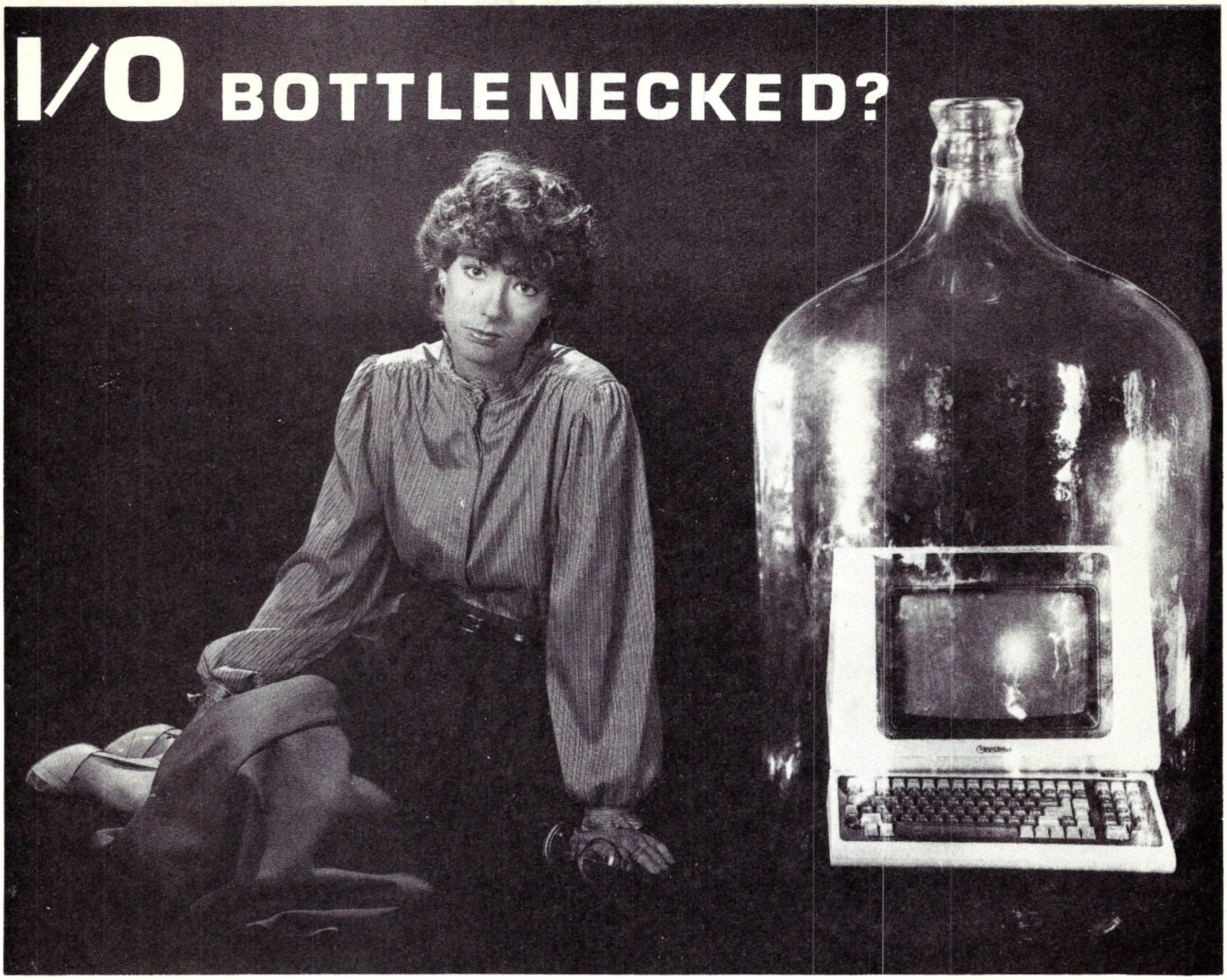
☐ Dealer ☐ Distributor ☐ Computer Manufacturer

NAME ____________________
COMPANY ____________________
ADDRESS ____________________
CITY ____________________
STATE __________ ZIP __________
PHONE ____________________

☐ Please have a salesman call
☐ Send literature only

InfoSoft
SYSTEMS INCORPORATED
25 SYLVAN ROAD SOUTH
WESTPORT, CONN. 06880
(203) 226-8937

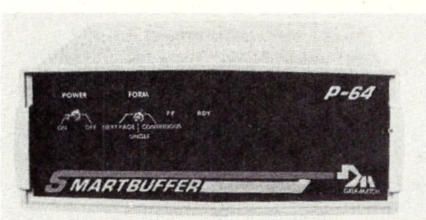
CIRCLE 7 ON READER SERVICE CARD



SMARTBUFFER

FREES PRINTER BOUND SYSTEMS

The Smartbuffer is an intelligent, buffered interface that rapidly accepts data from the computer or modem and then goes off-line to drive the printer. The computer is now free.



INTELLIGENT

CODE CONVERSION, SOFTWARE ROUTING, SINGLE SHEET, CHARACTER COMPRESSION

The Z80 microprocessor based system comes preprogrammed as a large buffer. Simply plug the SMARTBUFFER in between a modem or dedicated line and a terminal/printer or between a micro or mini and a modem or printer. The SMARTBUFFER also recognizes a series of commands which allows code conversion tables to be easily loaded, (i.e. EBCDIC to ASCII or ASCII to BAUDOT), remote steering of input files to different output ports, remote reset to clear the buffer and remote setting of USART parameters.

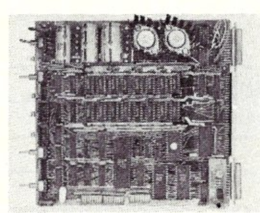
BUFFERED

Memory is user upgradable. Any 16k unit can be field upgraded to 32k by adding another row of 16k RAM chips. The same unit can be upgraded to 64k or 128k by replacing the 16k chips with 64k RAM chips. Only one type of RAM chips may be used on one card at a time.

INTERFACE

Serial to parallel, parallel to serial, baud rate, XON, XOFF or DTR, and character bit length conversion allows interfacing incompatible equipment. Multiport models allows more than 1 computer to be connected to more than 1 peripheral. Serial ports are each full duplex and are designed to accept external clocks for synchronous operation.

Based on the powerful and popular Z-80 microprocessor, 2716 ROM, 8251 USART and other standard TTL component, the Smartbuffer is easy to program and repair. A special command allows user programs to be easily downloaded into the Smartbuffer and execution turned over to this software. ROM I/O and memory management routines may be called as subroutines.



Break your I/O bottle necks with a Smartbuffer.

DATA-MATCH CORP.

3810 Oakcliff Industrial Court
Atlanta, Georgia 30340 • (404) 441-0408

CIRCLE 96 ON READER SERVICE CARD

The CP/M Bus

by Anthony Skjellum

Using C Instead of Assembly Language

Traditionally, system software and utilities for the CP/M operating system have been written in 8080 or Z80 assembly language. This was a natural starting point, since high-level language compilers did not exist for CP/M when it first appeared. Today, programmers may choose one of several such compilers instead of assembly language. The C programming language is an excellent choice for many types of utilities and reduces both development and debugging times relative to those for assembly language. Furthermore, the resulting software is more easily maintained and enchanced.

Familiarity with the C language will be assumed. For further information, the reader should refer to *The C Programming Language* by Kernighan and Ritchie, published by Prentice-Hall. The Nov/Dec 1981 issue of *Microsystems* (Vol. 2, No. 6) also contains a section on the C language that should prove instructive.

Parameter Manipulation

Most utility programs require one or more command line parameters. These parameters often include file names and/or flags to direct the details of execution. Parsing the command line can be a formidable job if a variable number of parameters is possible and the command line is not strictly of a fixed format. In C, simple command line processing is quite straightforward. The variables **argc** and **argv** allow direct access to preparsed strings. Alternatively, parsers may be created to provide extremely general command line processing (see my article, "Argum: a C Command Line Processor," in the May 1982 issue of *Dr. Dobb's Journal*). In either case, the programmer's task is much simpler than in assembly language.

Format Conversion

Format conversions are the bane of the assembly language programmer. The need for conversions is obvious: utility programs must deal with input data and produce output which often will contain data in a different form. In C, powerful input conversions and output formatting are available through the standard **scanf** and **printf** functions.

Debugging

The **printf** function also provides an effective means of debugging. The function may be used to display crucial information during execution. It is desirable to the same in assembly language, but it can be done far more readily in C.

String Manipulation

String manipulation in C acts either on a character array or on a string unit. Functions are provided to perform various types of string manipulation, including concatenation. This leads to programs that detect and handle illegal data intelligently.

File Access and Processing

The assembly language programmer is faced with further frustration when it comes time to access data from mass storage. Data must first be buffered for reading and writing. Macros do exist for handling buffered sequential input/output (e.g., SEQIO.LIB), so this process is not as difficult as it could be. However, file processing under C is especially convenient and transparent to the programmer. Most often, the program deals with the file name in its string format only; runtime routines handle conversion to FCB format.

Calculations

C provides operators to perform the same basic types of operation that are often performed in assembly language code. Included are logical AND, OR, and exclusive-OR. Furthermore, increment and decrement operators are available. Many of the available compilers perform some degree of code optimization so that C calculations are not excessive expensive in terms of execution speed.

Pointers

In 8080/Z80 assembly language, the 16-bit register pairs BC, DE, HL, SP (and index registers IX, IY in the Z80) may be used to point to locations within the 64K memory map. One can think of these registers as pointers in a low-level sense. In C, pointers ultimately refer to a specific type of variable, structure (a user-defined variable type) or union, and may have more than one level of indirection associated with them (e.g., a pointer to a pointer to an integer). Since an array name is itself a pointer, array and pointer references

may be intermixed for convenience. Finally, arithmetic on pointers takes into account the size of the object to which they point, so the programmer need not worry about this.

Program Structure

C is a block-structured language which encourages structured and modular programming. Programs are generally indented to indicate the level of block depth at any given point. When combined with explanatory comments, a C program will generally be more readable and maintainable than the comparable assembly language program.

Local Variables

In normal assembly language environments, all variables are global. That is, they are defined at assembly time and consume memory space throughout the execution of the program. In C, both global and local variables are supported. This permits temporary data to be exactly that: Once a piece of information is no longer needed, it may be discarded. Local variables are generally automatic. This means that they are allocated upon entry to a function and are deallocated upon exit from that function. A full C implementation permits static variables which retain their values between executions of the subprogram or function in which they are declared. Most subset C compilers do not support static variables. In any case, one does not normally use a static variable for temporary data, since it takes up space throughout execution of the complete program.

Dynamic Allocation

A second type of temporary storage is also available in many C implementations. This is dynamically allocated storage. This type of storage is requested through a special function call, **calloc**, which provides a pointer to a block of memory of the requested size, if possible. When the program has finished using the block, the storage may be deallocated for later use as part of another dynamically allocated block. The dynamic allocator handles all the specifics of the operation, and the program merely requests allocation and dealloca-

tion as needed. This type of storage can be accessed by various program levels merely by supplying the block's attributes (i.e., starting address and length). These may be kept in global variables or passed to a subroutine as a pair of parameters. If originally allocated by a subprogram, the storage does not automatically disappear upon exit from that subprogram; a specific deallocation request is required to free the space.

Global (External) Storage

External or global variables are analogous to variables defined in an assembly language program. They are available to the main program and to subprograms. As in assembly language programs, external variables contain flags and other information needed often enough throughout the program that it would be extremely inefficient to pass them as parameters to the various subprogram levels.

Redirection of Input/ Output

Many CP/M-based C implementations support an important feature derived from the Unix environment. This is the ability to redirect standard input and output streams (normally directed to the console input and output) at execution time to either disk files or other CP/M I/O devices. Another feature, called piping, allows a chain of modular programs to work on one stream of input data and produce a single output stream. Generally, assembly language programs will not support this feature unless they make use of a powerful software library such as that provided with Knowlogy's Unica/ XM-80 package.

Additional Comments

We have outlined some of the reasons that programmers should choose C over assembly language. It should be stated that in certain cases programs will require an execution speed that only directly coded assembly language can provide. In such cases, the programmer may be forced to use assembly code instead of C. Alternatively, the critical parts of the code may be written in assembly language and then linked to a

Coming Next Month

Ada Tutorial Part II by Mark Zeiger

Review of the JANUS Compiler by Harvey Fishman

Three More C Compiler Reviews by David Gewirtz

Comparison of CP/M and MSDOS by Steven Leibson

A Virtual Disk Facility for N* DOS by D. J. Anderson

Review of 26MB Hard Disk by Paul Earley

Turnkey + by James Offenbecher

And much more, including
our usual Departments and Columns

The CP/M Bus
continued . . .

main program written in C. For most applications, programmers will find that they are able to produce more intelligent software with better human engineering by using C.

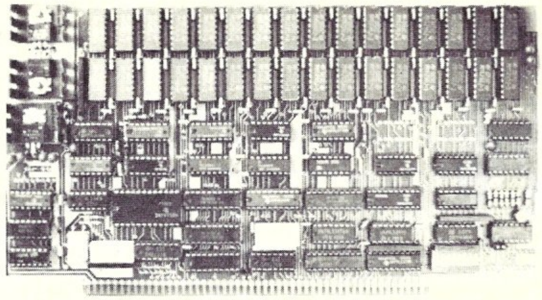
Conclusion

The C language provides a viable alternative to assembly language for many types of utility software to run under CP/M. It provides easier maintainability, shorter development and debugging times, and better error detection/reporting mechanisms.

The next several CP/M Bus columns will deal with code relocation. I shall be dealing with single modules that can be directly loaded to various boundaries, as well as modules to be linked together.

256K RAM IN 4K BLOCKS

1/4 Megabyte $1495



OVERVIEW

The BSR 64/256 is an 8 bit bank selectable dynamic random access memory card designed to operate in a Z-80 based S-100 computer system with a CPU clock frequency of up to 4 MHz(A model) or 6 MHz(B model).

Individual 64K banks are selected via the IEEE 696 8 bit address us extension. If the host system is not capable of driving the extended address bus, one of the BSR 64/256 cards in the system may be configured to drive it through an onboard latched output port.

System area is allocated in 4K blocks by writing a system mask out to two latched output ports. Another port allows any one of up to eight cards to be assigned as the current system master. Logically, up to 64 cards may be addressed in a single computer system.

Assembled & Tested Boards

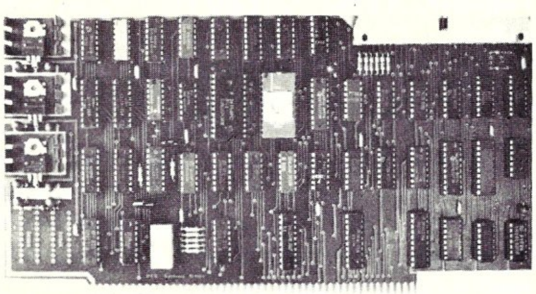
(4164) 64k $770
128k $1015
192k $1260
256k $1495

Memory prices fluctuate, call for price

FEATURES

MAX STORAGE	256k bytes (32 ea. 64k X 1 chips) or 64k bytes (32 ea. 16k X 1 chips)
BLOCK SELECTION	Any combination of 4k blocks in any 64k bank, software selectable
SYSTEM AREA	Any combination of 4k blocks in low order bank of current system master card, software selectable
SYSTEM MASTERS	Any one of up to 8 cards software selectable, one card jumper selectable for system power-up or reset
CARDS/SYSTEM	Logically, up to 64 64k or 256k cards
BANK SELECTION	Uses or implements IEEE 696 (S-100) extended address bus
CPU SPEED	4 MHz or 6 MHz with no wait states
REFRESH MODE	Invisible
OUTPUT PORTS	4 Consecutive ports for entire system, selectable on any 4 port boundary

14″ WINCHESTER FIXED DISK CONTROLLER & SUBSYSTEM



FDC 4000

14″ Winchester Fixed Disk Controller

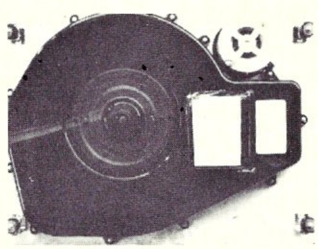
Controls Shugart 4000 & 4100 series drives
Up to 174 megabytes per controller
I/O mapped buffer - no wait states
Includes MP/M* XIOS & CP/M* CBIOS

Assembled & tested $895

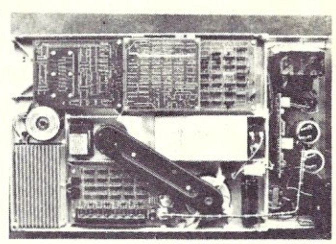
29 Megabyte Disk Subsystem $3840

The fixed disk subsystem includes a 29 million byte Shugart fixed disk (SA 4008), and an assembled and tested S-100 disk controller (FDC 4000). Also included is an unfinished rack mountable aluminum case, punched and ready for assembly. To complete this kit we have included power supply, fan, wiring harness, signal cable, and MP/M* XIOS and CP/M* CBIOS.

(SA 4008)



(Assembled Subsystem)



"Contact Us For More Information"

PCC SYSTEMS 5232 Manzanita Ave. Carmichael Ca 95608 (916) 338-5454

Price subject to change

*Trademark of Digital Research

The S-100 Bus

by David J. Hardy

Have you ever thought about bringing up MP/M on your machine? Or wanted to add some timers or a real-time clock? If your CPU board already has a CTC or two, then a few lines of code may solve your problem, but if you are one of those unfortunate souls (like me) who is without CTCs, you might like to try using an 8253 timer instead.

At first glance, a CTC would seem the best choice for a programmable timer, but a closer look reveals that the CTC requires some signals from the CPU IC itself, and that only a Z80 IC can be readily used. Besides, few CPU cards have enough leftover space to allow the kludge or wire-wrap installation of such a large IC.

The 8253, although just as large, does not require "special" (non-S100) signals from the CPU, and can be easily mounted on an S-100 wire-wrap card. There are a few limitations to the 8253, such as its 2MHz maximum clock frequency, and it isn't quite as versatile as the CTC, but (with maybe a bit of extra code) the 8253 can do most of the things the CTC can.

The 8253 contains 3 separate 16-bit timers, each with its own external clock input, control gate, and output. A single timer in the 8253 may be used to cause some external event, like sending a pulse to a printer, or turning on a LED, by simply connecting its output line (OUT1-3) to a proper TTL driver stage. In addition, a timer may be started and stopped by using its external gate input. A timer may also be used to generate interrupts by connecting its output to an open collector TTL buffer, and connecting the buffer's output to the S-100 bus INT* line (pin 73). INT* is an input to the CPU that can be pulled low by any S-100 card to cause an interrupt.

Each timer in the 8253 can be programmed in one of six different operating modes. These include the ability to make each timer look like a down-counter, a programmable one-shot, a rate generator, a square wave generator, a software-triggered strobe, and a hardware-triggered strobe. In addition, the timers can be made to count in binary or BCD. Complete programming information is available in the Intel 8253 product description, or the Intel Component Data Catalog.

David J. Hardy, 736 Notre Dame, Grosse Pointe MI 48230

The wire-wrap circuit shown in Figure 1 is simple and straightforward. I used the same decoding and bus interface circuits that Fred Deadrick used in his article about the MSM5832 clock ("An S-100 Clock Calendar Circuit," *Microsystems*, Jul/Aug 1981). ICI is a 6-bit comparator that enables the 8253 and its bus drivers IC2 and IC3 whenever the proper address is selected, and whenever the sINP or sOUT signals say that PORT type I/O (as opposed to MEMORY type I/O) is being done. When the proper address is selected, and PORT type I/O is determined, pWR* and pDBIN tell the bus drivers and the 8253 to read from or write to the bus. The 8253 itself decodes two more address lines to find out which of its four ports is actually being selected.

It may sound complicated, but it's actually very simple, and it all happens in less than a few microseconds in a typical S-100 system. The 8253's clock inputs can be from any TTL source, but the maximum input frequency must be 2MHz or less. The S-100 bus conveniently provides a 2MHz clock line (pin 49) that is used here to drive the clock input of each of the 8253's three timers. However, a timer could also be used as, say, an event counter, by connecting the event signal to its clock input.

One thing that the 8253 can't provide as easily as the CTC is vectored interrupts. If you use the 8253 to generate interrupts via the INT* line, it will only be able to generate a simple maskable interrupt request. In most systems, this would be equivalent to an RST 7 instruction, which would cause the processor to push the PC onto the stack, and then jump to address 38H.

I say most systems, because this type of interrupt actually just causes the CPU to expect some external device to cram an instruction onto the data bus for it to interpret. A vectored interrupt system, for example, might place a jump (OC3H) instruction onto the bus, except that a jump requires two or three bytes. The RST instructions were invented just for this reason. They require only a single byte, and so they can be crammed onto the data bus during an interrupt, and cause a jump to an address in page zero of memory. Since there is no logic here to cram anything onto the bus when an interrupt occurs, the CPU (assuming the data lines are not being pulled down by some other board) will "see" all one's on its data lines

during an interrupt. All one's (i.e. 0FFH) just happens to be the code for RST 7. Clever, those folks at Intel. . . .

The Z80 CPU is a bit safer to use this way than an 8080 or 8085, because when used in interrupt MODE 1, it will always "see" an RST 7 during an interrupt request.

The program in Listing 1 is a simple test and demonstration program that should give some idea of the general coding required to use the 8253 as a timer. Basically, it sets up one of the 8253's timers to generate an interrupt, then reports to the console if that interrupt occurs. The same basic operations used in this program can be used to implement interrupts in an MP/M system, or provide programmable time delays.

This column is the first of what I expect will be a regular column in *Microsystems*. The column will serve as a forum on S-100 topics. I encourage readers to send in any questions about the S-100 bus, which I will attempt to answer in this column. The questions should, in general, be directly related to the hardware structure and timing of the bus, though some software may also be involved. Questions could be general ones about interfacing to the S-100 bus, or specific ones about problems encountered in trying to interface a specific product. Until some questions are received, here is an S-100 circuit that you may find interesting.

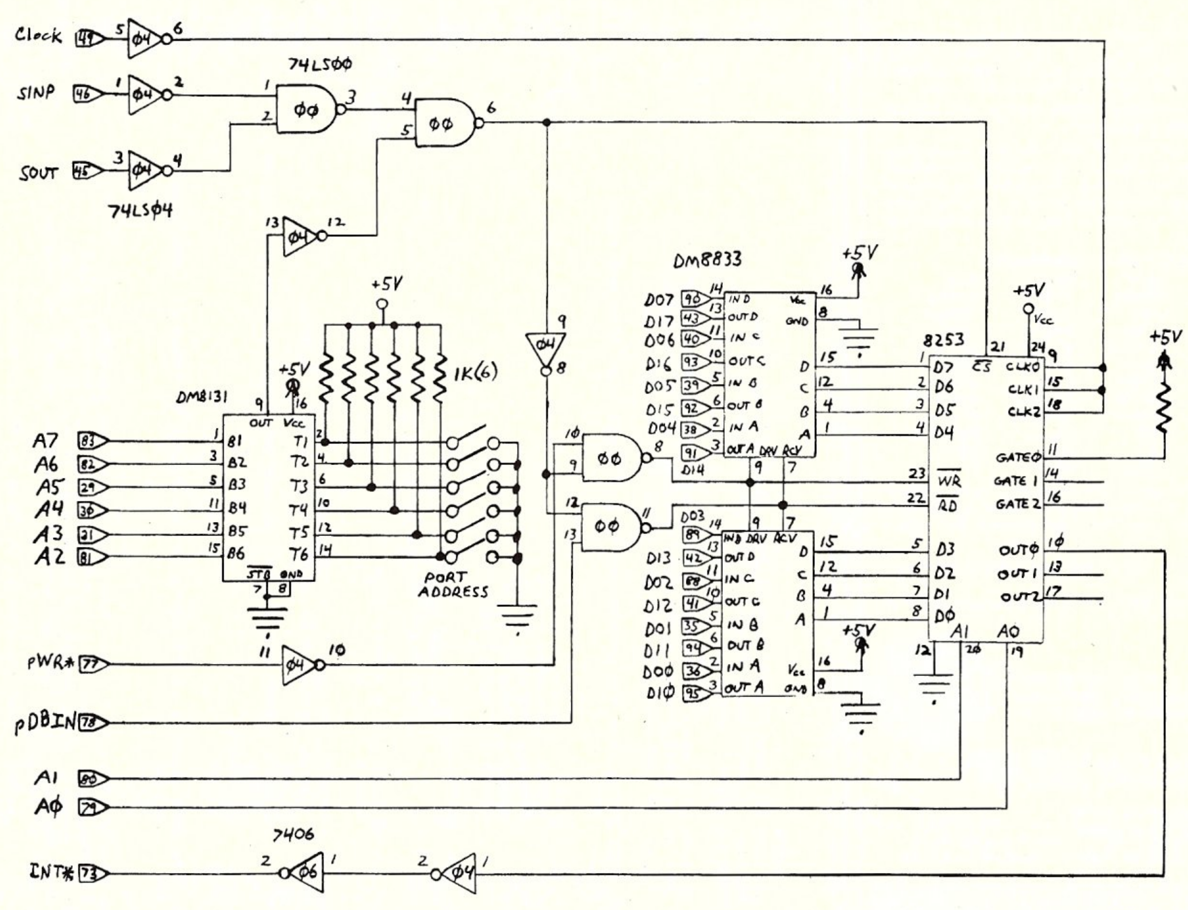


Figure 1

```
;
;   IGEN PROGRAM
;        sets up and tests 8253 counter #0 to see if interrupt
;        hardware is working.  Prints a message telling if it
;        does or doesn't work.
;
;   This program will send the message "Interrupting" to the console
;   each time an interrupt request is generated by the 8253.  If the
;   8253 circuit is working properly, ten interrupts should occur.
;
;   Define 8253 ports base (as determined by the comparator IC's switches)
PBASE   EQU     70H
;
;   Define 8253 ports
CTR0    EQU     PBASE   ;Counter 0 I/O
CTR1    EQU     PBASE+1 ;Counter 1 I/O
CTR2    EQU     PBASE+2 ;Counter 2 I/O
CTRL    EQU     PBASE+3 ;Control Port
;
;   Define some BDOS values
BDOS    EQU     5       ;BDOS JMP address
PRINT   EQU     9       ;BDOS Print String function
;
;   Set origin to 100H for standard CP/M
        ORG     0100H
;
;   Set up RST 7 handler in page 0, address 38H
        MVI     A,0F3H  ;Store a DI instruction at RST 7 address
        STA     0038H
        MVI     A,0C3H  ;Then store a JMP to the interrupt handler subroutine
        STA     0039H
        LXI     H,PRNTIT
        SHLD    003AH
;
;   Initialize pass counter to 10 (this is the number of times to interrupt)
        MVI     A,0AH
        STA     1000H
;
;   Set up Counter 0 to as a rate generator for a 17mS countdown
;   This will make the counter run forever issuing a pulse every 17mS
;   (Note that another popular way to do this is to set up the counter
;   as a down-counter, then have the interrupt handler reload the counter
;   after each interrupt.)
INT     MVI     A,34H   ;Select Counter 0, Mode 2, Binary decrement
        OUT     CTRL
        MVI     A,34H   ;Load down-counter with 8234H for 17mS @ 2MHz
        OUT     CTR0
        MVI     A,82H
        OUT     CTR0    ;Counting starts automatically after this OUT is done
;
NOTYET  EI              ;Enable interrupts
;
;   Now wait for an interrupt to happen...
        LXI     D,WAITMSG       ;Say that we're waiting...
        MVI     C,PRINT
        CALL    BDOS
;
;   Waste time forever, checking if done, until an interrupt occurs...
MORE    LDA     1000H   ;Check pass counter
        CPI     0       ;If not zero, then continue
        JNZ     MORE    ;Jump if more passes to go...
        DI              ;Else disable interrupts (counter is still going...)
        LXI     D,RETMSG        ;Say that we're returning to CP/M...
        MVI     C,PRINT
        CALL    BDOS
        JMP     0       ;Then return to CP/M via a warm-boot
;
;   Interrupt routine to tell us when an interrupt has occurred
PRNTIT  PUSH    PSW     ;Save all registers so they will be unchanged
        PUSH    H       ;  after we return from interrupt
        PUSH    D
        PUSH    B
        LXI     D,INTMSG        ;Say that interrupt has just happened
        MVI     C,PRINT
        CALL    BDOS
        LDA     1000H   ;Decrement pass counter
        DCR     A
        STA     1000H
        POP     B       ;Restore all registers to their original states
        POP     D
        POP     H
        POP     PSW
        EI              ;Re-enable interrupts for next time
        RET
;
;   Status messages
;
WAITMSG DB      'Waiting for interrupts...',0DH,0AH,'$'
INTMSG  DB      'Interrupting',0DH,0AH,'$'
RETMSG  DB      'Returning to CP/M...$'
;
        END
```

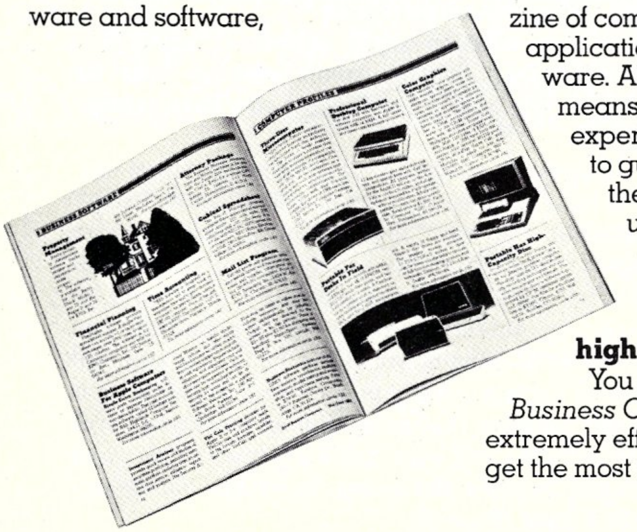
Are you getting the most from your business computer?

Get the best source of information available on microcomputer use: *Small Business Computers*

It pays to consult *Small Business Computers* before you or your company expand your software capabilities, look at new peripherals, or even consider your first office microcomputer.
Here's why:

We write only for people who are professionally involved with business microcomputer systems. You won't find any space games or home budget programs in our pages.

Instead, you *will* find feature after feature on business hardware and software,



plus applications you can use right now, and later as your needs grow.

SMALL BUSINESS COMPUTERS critically reviews a variety of software packages for:

- Accounting
- Word processing
- Data base management
- Business planning
- Financial modeling
- Spreadsheet analysis
- File and record handling

—and much more, from list management to project control.

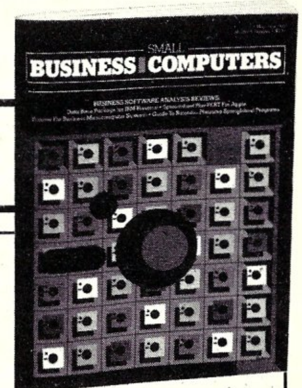
You'll also evaluate a wide range of hardware and peripherals, including video display terminals, disk storage equipment, printers (dot matrix and daisy wheel) and modems.

Every evaluation and review in *Small Business Computers* is conducted by our own panel of 21 top computer-science experts. What's more, *Small Business Computers* comes to you from the publishers of *Creative Computing*, the Number One magazine of computer applications and software. All of which means we have the expertise and resources to guarantee that you get the most authoritative, up-to-date analyses and advice available anywhere.

Cost effectiveness: high value plus low price

You can see why *Small Business Computers* will be extremely effective in helping you get the most from a business computer system. Now for the "cost" part: You can subscribe to *Small Business Computers* for as little as $14.97 and save 25% off the full 12-issue subscription rate. Or take 24 or 36 issues and save up to 35%!

Whatever your level of involvement with business microcomputers, it makes sense to get the best source of practical information around. Why not take a moment to fill in and mail the coupon now?



Your subscription to _SMALL BUSINESS COMPUTERS_ may be tax deductible. Check with your accountant.

SMALL BUSINESS COMPUTERS CN 1988, Morristown, New Jersey 07960

YES! Enter my subscription to **SMALL BUSINESS COMPUTERS** for:

- ☐ 12 issues at $14.97—I save 25%.
- ☐ 24 issues at $27.97—I save 30%.
- ☐ 36 issues at $38.97—I save 35%.

Savings based on full 12-issue subscription rate of $19.97.
Offer valid in U.S. and possessions only. Please allow 30 to 60 days for delivery of first issue.

CHECK ONE:
- ☐ Payment enclosed.
- ☐ Purchase order enclosed.
- ☐ Bill me later.

8H04

NAME________________________
(please print full name)

COMPANY________________________

ADDRESS________________________

CITY____________ STATE____________ ZIP____________

Innovations in High-Level Language

A concise history of computer languages that have led to the current state of affairs, and where we are going

by Don Libes

Are you dismayed with the power of your current programming language? Can you think of "features" that would enhance its power or enable you to get your job done more quickly? If your ideas are applicable for a broad range of uses, you may have the ability to design a new high-level language. As proof that one person can do it alone, consider Niklaus Wirth who designed Pascal, and Ken Iverson who designed APL. In most other cases, successful languages are designed by committee, but you can do it yourself, simply by capitalizing on the work of predecessors in language design and choosing what things will work well together in a high-level language. I am not going to claim that it is really possible for the amateur to be successful in language design, but an awful lot have tried.

It is estimated that there are over 150 high-level languages in use today (perhaps in the Department of Defense alone). Doubtless, hundreds of others have been used in the past, and hundreds more have been proposed without implementation.

For most people who communicate algorithms with computers, the languages of importance are the so-called "high-level" languages. There is such a surprising disparity in the languages that are known as high-level that it is virtually impossible to explain the term without admitting an obviously deficient language such as a macroassembler, or excluding one as powerful as Algol (which can be interpreted directly on several Burroughs computers).

Rather than rigorously defining the term, Jean Sammet, an authority on programming languages, suggests the following characteristics of high-level languages:

(1) The language does not require the user to have any knowledge of machine code or machine characteristics.

(2) The language is inherently independent of any particular computer, and hence it should be easy to take a program written in a high-level language and run it on many machines.

(3) There is usually a one-to-many translation from executable elements in the source code to specific machine instructions.

(4) The notation of the high-level language is fairly natural, relative to the problem area for which it is intended and it is not in a fixed tabular format.

Historically, programming languages have never satisfied everyone in being able to state "their" algorithm the way "they" think is easiest. More important is the ability to express an efficient algorithm. Efficiency, unfortunately, can refer to time, program length, space (off-line, on-line) and a number of other interests. In the face of physical reality, these naturally present real tradeoffs. It is understandable that no computer service has ever stopped tinkering with its billing algorithms in an attempt to solve the non-truth of how many seconds a byte is worth.

The best language is always around the corner. Languages have come and gone with satisfying alacrity. I do not hail any one language as being the best of all, no matter what kind of snazzy runtime feature I am provided with. With the satisfaction that we have advanced a giant step in our knowledge, I quote from page 2 of the Preliminary Report (1954) on Fortran: " . . . FORTRAN should virtually eliminate coding and debugging. . . ."

It is worth examining some of the good ideas that particular languages have given us. These are the quirks and, in some cases, the leaps in thinking that languages have made for us. These are the

Don Libes, 60 Gold Street, Rochester
NY 14627

ideas that will affect future language designs.

I will attempt a chronological order of presentation. There is an inevitable blurring for languages such as Snobol which was reintroduced several times (i.e. Snobol, Snobol2, Snobol3, and Snobol4). Not all of the important ideas appeared in first versions; nonetheless, I tend to associate the first appearance of a language with the advances and features that characterize it.

Fortran

The first language to become widely implemented, Fortran gets the credit for many ideas in programming languages. I am continually astounded, however, that it is still currently in use. While I can't admit that the claim holds nowadays, Fortran was once an extremely efficient language. It had to be. It had to prove to many nonbelievers that programming could be "automated" efficiently. Naturally, there was (and remains) a sacrifice from the high-level language point of view, but Fortran broke new ground in showing that producing an efficient program did not require ingenious hand coding (except for the original implementations!).

Cobol

Perhaps the most widely implemented language (because of its connections in business!), Cobol was the first language that demonstrated "portability" through standardization. The importance of portability is a lesson we have still not learned, as is evident from the many versions of Basic which adhere to no standard and make portability a real problem.

Cobol also showed us that just because people can "read" programs, does not mean they "understand" them. We must admire, though, Cobol's English-like style in producing language forms such as the following:

```
IF A = 2, 3, or 4 . . .
```

and

```
IF X = 1 ADD 1 TO Z
IF GREATER ADD 2 TO Z
OTHERWISE . . .
```

Cobol also caused the maturing of many different types of file access techniques.

Algol 60

Algol was the first attempt to create a universal computer language. A benchmark in language theory, it has been the basis of many later computer languages, including Pascal, Simula, PL/I, Bliss, C, Sail and many others. Algol's most important contribution to high-level languages was its notions of block structure, including nesting, compound statements, procedures and scoping. Notable also for its accidental achievements, Algol allowed recursion and parameter passing by name ("thunks").

The use of BNF (Backus-Naur Form) in defining the syntax was a giant step and has been used for programming languages ever since. It is unfortunate that there is nothing as appropriate for defining language semantics.

Algol took an attitude towards I/O that was exactly opposite to that of Cobol. I/O was defined not to be a part of the language proper. This, unfortunately, may have led to Algol's not being accepted as a practical language. Instead, it has remained primarily a language for the communication of algorithms.

Lisp

The premier language of the artificial intelligence community, Lisp pioneered computation with symbolic expressions rather than numbers. Using recursion and its fundamental datatype, a "list" (alias the binary tree), Lisp has enabled us to describe anything in mathematically elegant ways. (Of course, some people don't particularly care about elegance!)

Lisp also initiated the study of garbage collection as a means of memory allocation, and passing functions as parameters.

Simula

Based on Algol, this simulation language introduced fundamental ideas on data abstraction (objects, messages and classes). Simula provided the first high-level tools for handling synchronization and manipulation of processes as actual data in the language (naming, declaring, coroutining, etc.).

Joss

The first language designed specifically for an interactive environment (and timeshared besides!), Joss faced (and solved) the initial problems of friendly user interfaces. Though no more complicated than Basic, Joss was easy to learn and use, and it completely shielded the user from the harsh

Languages have come and gone with satisfying alacrity . . .The best language is always around the corner.

vagaries of the computer that it ran on. It took special care to avoid anomalous conditions and it gave clear error messages.

The first remote terminal, designed especially for Joss, used a two-color ribbon so that the user typed in green while the computer responses were in black! (With bitmapped screens, it is only 20 years later that we see this old idea of disambiguating roles being reintroduced!)

Snobol

A "socially unacceptable" language because of its lack of modern control structures, Snobol thrives nonetheless because of the power it provides in performing string manipulation, one way of looking at all computations. Snobol's most ingenious concept is that of the "pattern" datatype (which I don't have the space to give justice to here).

Snobol treats strings as scalars in contrast to other languages that handle strings as arrays of characters. Snobol "tables" are a natural consequence of this idea, allowing indexing by strings (intuitively akin to maintaining a symbol table). For example:

```
color<'elephant'> = 'grey'
```

The language is absolutely full of wild and incredible ideas. Heterogeneous arrays, delayed execution of expressions, and run-time compilation made their first appearance in Snobol.

One must also wonder at the internal consistency of the language—the first usable implementation (1963) of Snobol took "about three weeks," demonstrating that high-level language design does not necessarily have to be such a very costly undertaking.

PL/I

"If it was felt that a facility was useful and could be compiled . . . it was added to the language." This statement from George Radin (IBM) in a paper on the early history of PL/I explains why many have called the language a "100-bladed Swiss knife."

This language showed us that having "everything" was not the solution. A representative of what can go wrong when you "design by committee," PL/I had everything including a myriad of datatypes and their implicit conversions, the interactions of which surely accounted for much of its apparent complexity.

The official manuals for PL/I were thick and dense, but if you were the type of person who loved to impress your friends with really arcane knowledge, you could find such gems as the 26 ON-conditions for interrupts (surely the nemesis of any optimizing compiler) or the Sterling declaration should you quickly need to determine the total of your British and US assets!

Basic

I'm sorry, all you Basic fans. There just wasn't anything new about Basic. It was a combination of several languages, particularly Joss and Fortran.

APL

As with Lisp and Snobol, APL views the world in its own consistent way, via the array. By choosing the array as the primitive datum, APL functions can internalize the control structure of an APL, thereby allowing the language to be interpreted efficiently (which is fortunate since it is difficult to compile).

APL makes use of an unusual character set and an unusual rule for precedence in the face of complicated choices of precedence and associativity for primitive functions (e.g. Snobol has 15 levels of precedence). APL's simple rule is that execution is from right to left except when overridden by parenthesis.

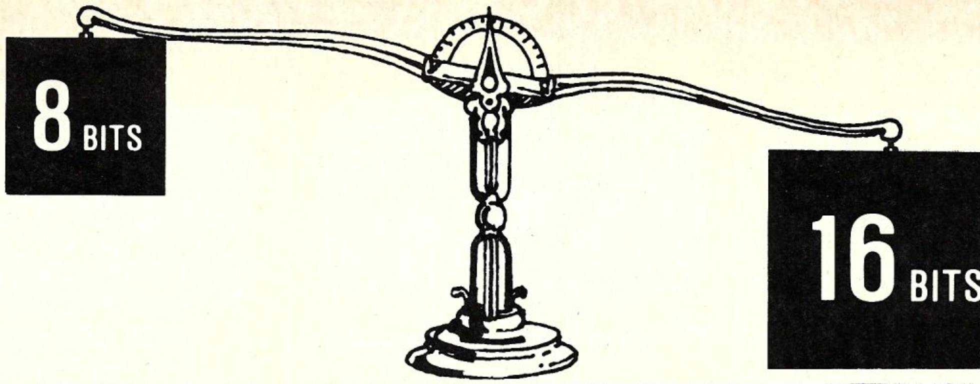
Smalltalk

With heavy influence from Simula, Smalltalk provides a consistent treatment of the programming environment in an object-oriented system. Not only are the classes of Simula objects in Smalltalk, but so are all data structures and control structures. Smalltalk also expresses relations between classes and includes ideas such as metaclasses (classes that describe classes), and inheritance of classes.

Pascal

Based primarily on Algol, Pascal was designed to be a language suitable for teaching, efficient execution and reliability, the last of which it innov-

Shakespeare on CP/M compilers:
"They have been to a great feast of languages and have stolen the crumbs."
(Love's Labours Lost, Act V, Sc. I)



MICROCOMPUTERS

THE CHOICE JUST GOT EASIER!

PRICE $4390 Gets you a Tec-86 Complete 16 Bit 8086 System, including 64K Bytes of Memory, Dual 8 Inch Double Density Floppy Disks, and more. . . (See complete Tec-86 specs below.)

PERFORMANCE Faster Processing, Increased Address Space, up to One Megabyte of available memory, and the Increased throughput of the the TEC-86 16 Bit System means that you can now do work on a microcomputer never before practical with the slower 8 Bit systems.

SOFTWARE CP/M-86™ Brings the World of CP/M software up to the 16 bit systems. BASIC-86™, MS-Fortran™, MS-PASCAL™ are just a few of the Languages currently available.

EXPANSION Memory Expansion up to ONE MEGABYTE. Disk Storage can be increased to over 30 Megabytes with our Winchester Disk Subsystem. Multi User Systems available soon!

QUALITY Full ONE YEAR WARRANTY is standard on TECMAR components. This Warranty is possible due to our High Engineering Standards, and Thorough Product Testing.

DELIVERY We have been delivering 16 Bit Boards and Systems for nearly two years. Most orders are shipped from stock or within 30 days.

S100 A/D $765
Analog to Digital Converter with Timer/Counter

12 Bit Accuracy, 16 Channels, 30 KHz Conversion, Five Programmable Timer/Counters for:
- Time of Day
- Triggering Conversions
- Counting Conversions
- Counting External Signals
- Interrupting the CPU

Options: Programmable Gain; 14, 16 Bit Accuracy; 40, 100, 125 KHz Conversion; Expansion to 256 Channels

$4390 TEC-86 $4390
16 BIT 8086 MICROCOMPUTER SYSTEM

FEATURES: 5 MHz 8086 CPU * 64K Bytes RAM * S100 IEEE-696 * Dual 8 Inch Double Density Floppy Disk Drives * Two RS 232 Serial Ports, 50 to 19200 Baud * Three Parallel Ports * ROM Boot for CP/M-86 * Heavy Duty Power Supply * Attractive Industrial Quality Desk Top Cabinet * Completely Assembled and Tested * **OPTIONS:** Up to One Megabyte of Memory, Double Sided Double Density Floppy Disks * Up to 31 Megabytes of Winchester Fixed Disk Storage. * CP/M-86 * BASIC-86 * MS-FORTRAN * MS-PASCAL * CBASIC/86™ * PASCAL/M86™

$10990 TEC-86W $10990

Same Great Features as our TEC-86 except that the TEC-86W has 256K of RAM, and our 31 Megabyte Winchester Disk Subsystem.

NEW — 64K/256K S100 MEMORY $595/$1695

8 and 16 Bit Transfer for 8 and 16 Bit Systems
MP/M and MP/M-86 Compatible, Up to 7 users on one 256K board.
Parity Detection with Interrupt and LED Indication
Hardware Write Protect
Full DMA Capability
24/20/16 bit Address Decoding
Enable/Disable 8K/32K Segments
Bank Select in 8K/32K Segments
Meets IEEE-696 Specifications

See our catalog for many other fine S-100 and Apple Boards including: — Analog to Digital Converters *(16-256 channels; 12, 14, or 16 bit accuracy; 30, 40, 100, or 125 KHz; programmable gain; timer/counters)* **— Digital to Analog Converters** *(12 bit accuracy, 3 microsecond conversion)* **— 8086 CPU Board — I/O Boards — 16K Static RAM — Real-time Video Digitizer and Display.**

TECMAR INC.

DEALER INQUIRIES INVITED

23600 Mercantile Rd. • Cleveland, OH 44122

TECMAR, INC.
(216) 464-7410

CIRCLE 33 ON READER SERVICE CARD

PRIORITY ONE ELECTRONICS

☆☆☆ CompuPro™ ☆☆☆

1 ONE

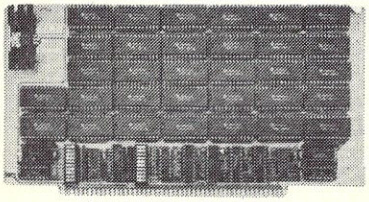
5 REASONS WHY YOU SHOULD SPEND AN EXTRA 5¢ WITH PRIORITY ONE

Some of you may have noticed that some other vendors have started to advertise CompuPro products. Frankly, we're amused that it has taken them this long to see that you want state-of-the-art IEEE/696 S-100 products. The discriminating user wants boards that meet the IEEE standard, not just any board with 100 pins that is obsolete before it is sold. We've known this for years; that is why we have continued to grow while others have had to cut back. To attract your business, they try to copy us and cut the price, often by as little as five cents. Rather than cut price, we offer the best combination of price, performance, and support in the business. We at Priority One are proud of what we offer for that extra five cents. Priority One Electronics is more than just a computer store or mail order house — we're an Authorized CompuPro Systems Center. Priority One is the first place you should turn to for business, scientific, and industrial computing solutions. Here are just five examples of what that extra five cents buys:

- As an Authorized Systems Center, we provide custom system integration from board level components
- We maintain full documentation on all CompuPro hardware and software
- We attend and hold periodic factory training seminars on CompuPro products
- We have factory trained Sales Engineers available to answer your questions after the sale is made. When you call Priority One, you'll speak with a professional, not an answering machine or answering service
- We buy directly from the factory, not from two step distributors, so we're always first to receive new products and new information about enhancements for existing products. When we buy direct, we we buy in volume. Whether you need one board or one hundred, we can usually deliver from stock

When you're ready for professional level, state-of-the-art micro-computing, turn to Priority One, the world's largest Authorized CompuPro Systems Center. We're ready for you.

☆☆☆☆☆☆☆☆☆☆☆☆☆



CMOS STATIC RAM

For a complete analysis of the advantages of CMOS memory, see the "Product Description" on page 416 of the January Issue of **BYTE**

RAM 17 - 64K CMOS STATIC RAM

RAM 17, 10 MHz, 2 Watt, DMA Compatible 24 Bit Addressing

Part No.	Description	List Price	Our Price
SIGBT175A64	64K A&T	$599.00	$550.00
SIGBT175C64	64K CSC	$750.00	$699.00

RAM 16 - 32K x 16 BIT CMOS STATIC RAM

8 and/or 16 Bit

(816) RAM 16 10 MHz, 32K x 16 or 64K x 8

IEEE/696 16 Bit 2 Watt, 24 Bit Addressing

SIGBT180A	64K A&T	$650.00	$599.00
SIGBT180C	64K CSC	$750.00	$699.00

(816) NEW! RAM 21 - 128K STATIC RAM

RAM 21 12MHz, 128K x 8 or 64K x 16 **WOW!**

IEEE/696 8 or 16 Bit 1.2 Amps, 24 Bit Addressing

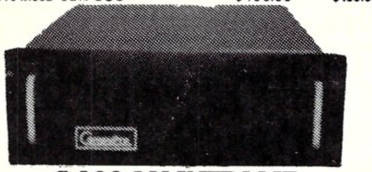
SIGBT190A	128K A&T	$1695.00	$995.00
SIGBT190C	128K CSC	$1895.00	$1795.00

STATIC MEMORY BOARDS

RAM 20 - 32K STATIC RAM

RAM 20 10 MHz, 4K byte block disable, bank select or 24 bit addressing available 8, 16, 24 or 32K

SIGBT164AA8	8K A&T	$210.00	$190.00
SIGBT164AC8	8K CSC	$280.00	$260.00
SIGBT164AA16	16K A&T	$285.00	$260.00
SIGBT164AC16	16K CSC	$355.00	$325.00
SIGBT164AA24	24K A&T	$355.00	$325.00
SIGBT164AC24	24K CSC	$425.00	$385.00
SIGBT164AA32	32K A&T	$425.00	$385.00
SIGBT164AC32	32K CSC	$495.00	$450.00

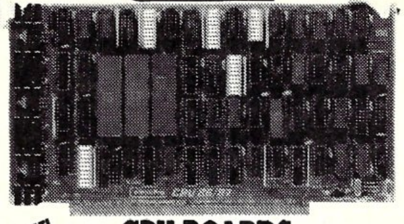


S-100 MAINFRAME

110V 60Hz CVT Mainframe uses famous 20 slot COMPUPRO Motherboard. 55 lbs.

SIGBTENC20RM	20 Slot Rack Mount	$895.00	$825.00
SIGBTENC20DK	20 Slot Desk Top	$825.00	$760.00

CompuPro™



NEW!

CPU BOARDS

CO-PROCESSOR 8086/8087 (816)

16 bit 8 or 10 MHz 8086 CPU with sockers for 8087 and 80130

Part No.	Description	List Price	Our Price
SIGBT186A	A&T 8MHz 8086 only	$695.00	$ 625.00
SIGBT186C	CSC 10MHz 8086 only	$850.00	$765.00
SIGBT186A87	A&T with 8087 option*	$1295.00	$1225.00
SIGBT186C87	CSC with 8087 option*	$1550.00	$1456.00

**8087 Limits clock speed to 5MHz*

DUAL PROCESSOR 8085-8088

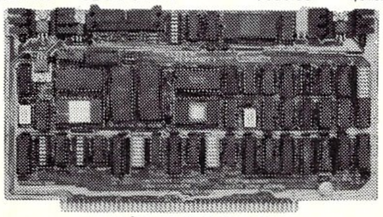
6 or 8 MHz Provides true 16 Bit Power with a standard 8 bit S-100 bus

SIGBT1612A	A&T 6Mhz	$425.00	$399.00
SIGBT1612C	CSC 6/8 MHz	$525.00	$498.00

CPU-Z Z80 CPU

2/4 MHz Z80 CPU 24 Bit Addressing

SIGBT160A	A&T	$295.00	$280.00
SIGBT160C	CSC 3-6 MHz	$395.00	$375.00



I/O BOARDS

SYSTEM SUPPORT 1 MULTIFUNCTION BOARD

Serial port (software prog. baud), 4K EPROM or RAM provision, 15 levels of interrupt, real time clock, optional math processor

Part No.	Description	List Price	Our Price
SIGBT162A	Assembled & Tested	$399.00	$360.00
SIGBT162C	CSC	$495.00	$460.00
SIGBT8231	Math Chip		$195.00
SIGBT8232	Math Chip		$195.00
SIGBT162AM1	A&T with 8231 Math Chip		$555.00
SIGBT162CM1	CSC w/8231 Math Chip		$655.00
SIGBT162AM2	A&T w/8232 Math Chip		$555.00
SIGBT162CM2	CSC w/8232 Math Chip		$655.00

MPX CHANNEL BOARD

I/O Multiplexer, using 8085A-2 CPU on board with 4K RAM

SIGBT166A4	Assembled & Tested	$495.00	$445.00
SIGBT166C4	CSC	$595.00	$535.00

With 16K RAM

SIGBT166A16	Assembled & Tested	$649.00	$585.00
SIGBT166C16	CSC	$749.00	$675.00

INTERFACER 1

Two Serial I/O

SIGBT133A	Assembled & Tested	$249.00	$219.00
SIGBT133C	CSC	$324.00	$298.00

INTERFACER 2

Three parallel, one serial I/O board

SIGBT150A	Assembled & Tested	$249.00	$219.00
SIGBT150C	CSC	$324.00	$289.00

INTERFACER 3

Eight channel multi-use serial I/O board

SIGBT1748A	Assembled & Tested	$699.00	$629.00
SIGBT1748C	CSC 200 hr. 8 Port	$849.00	$750.00
SIGBT1745A	Assembled & Tested	$599.00	$559.00
SIGBT1745C	CSC 200hr. 5 port	$699.00	$629.00

NEW!

INTERFACER 4

Three Serial, 1 Parallel, 1 Centronics Parallel

SIGBT187A	Assembled & Tested	$350.00	$315.00
SIGBT187C	CSC	$450.00	$415.00

SPECTRUM COLOR GRAPHICS

Color Graphics board with Parallel I/O

SIGBT144A	Assembled & Tested	$399.00	$349.00
SIGBT144C	CSC	$449.00	$339.00
SIGBT2D	Sublogic Universal Graphics Interpreter Software		$35.00

S-100 MOTHERBOARDS

Active termination, 6-12-20 Slot

SIGBT153A	A&T 6 slot, 2 lbs.	$140.00	$126.00
SIGBT153C	CSC 6 slot, 2 lbs.	$190.00	$175.00
SIGBT154A	A&T 12 slot, 3 lbs.	$175.00	$155.00
SIGBT154C	CSC 12 slot, 3 lbs.	$240.00	$220.00
SIGBT155A	A&T 20 slot, 4 lbs.	$265.00	$235.00
SIGBT155C	CSC 20 slot, 4 lbs.	$340.00	$310.00

ACTIVE TERMINATOR

Allows older S-100 motherboards to run faster and quieter

SIGBT106A	Assembled & Tested		$59.50

CompuPro

DISK CONTROLLERS

DISK 1 FLOPPY CONTROLLER

Fast DMA, Soft Sector, Controls 8" or 5¼" Single or Double Density. OUR BEST!

SIPDBT171ACPM	A&T w/CP/M2.2 & BIOS	$670.00	$450.00
SIGBT171C	CSC	$595.00	$555.00
SIGBTCPM80*	CP/M 2.2 for Z80/8085 with manuals & BIOS 8" S/D disk		$175.00
SIGBTCPM86	CP/M for 8086 with manuals & BIOS 8" S/D disk		$300.00

NEW!

DISK 2/SELECTOR CHANNEL HARD DISK CONTROLLER

Fast DMA 2 board set.Controls 4 Shugart 4000 series or Fujitsu 2300 type drives. Includes CP/M 2.2®

SIGBT177A	Assembled & Tested	$795.00	$750.00
SIGBT177C	CSC	$895.00	$850.00

Hayes™

SIDCH0200P	Smartmodem (Sh. Wt. 6 lbs.)	$279.00	$225.00
SIDCH0300P	Chronograph (Sh. Wt. 3 lbs.)	$249.00	$205.00
SIDCH0100P	Micromodem 100 (Sh. Wt. 4 lbs.)	$399.00	$325.00
SIDCH2901D	CP/M® 8" Terminal Program for above		$ 25.00
SIDCH0000P	Micromodem II	$379.00	$299.00
SIDCH2401D	Datacom (PASCAL patch) for above		$ 50.00

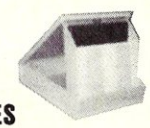
5" DISKETTES!

SOFT SECTOR

40 TRACK SINGLE SIDED DOUBLE DENSITY WITH HUB REINFORCING RINGS

PKG. OF 10 $19.95

BONUS!



FREE! KASSETTE 10 LIBRARY CASE WITH PACKAGE OF 10 DISKETTES

A $4.25 VALUE **Part No. BHPRI5SDD**

SIPRI580	package of 80 less Library Case	$120.00
SIPRI8SD	package of 10 8" S.S./D.D. Diskettes no sleeves	$20.00
5 or more packages		$18.00

SANYO

FCC CLASS 2 APPROVED DATA DISPLAY MONITORS

SPECIFICATIONS:

Viewing Screen	12" diagonal; 75 square inches DM2112: P31 phosphor
Scanning System	525 lines; 60 fields/second; overscan
Horizontal Resolution	600 lines, center
Signal Input	1.0 volt p-p composite video; 75 ohms

BHSYODM2112

List Price: $160.00 **$SALE: $119.00**

OTHER SANYO MONITORS ON SALE TOO!!

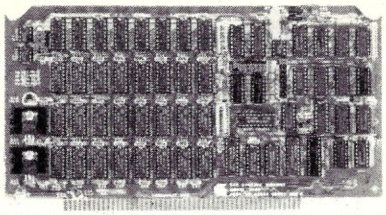
Part No.	Description	List Price	SALE
SISYOVM4509	9"B&W P4, 10MHz (15 lbs.)	$215.00	$149.00
SISYODM4509CX	9"Green, P31, 10MHz (15 lbs.)	$200.00	$159.00
SISYODM8012C	12"B&W P4, 18MHz (24 lbs.)	$250.00	$195.00
SISYODM8112CS	12"GReen, P31, 18MHz (24 lbs.)	$260.00	$199.00
SISYOVMC6013	13"Color, 16 x 64 (35 lbs.)	$470.00	$375.00
SISYODM6113*	13"RGB Color (35 lbs.)	$995.00	$895.00

**As used with IBM P.C.*

PRIORITY ONE ELECTRONICS

California Computer Systems

SPECIAL PURCHASE!!



2065 64K S-100 DYNAMIC RAM

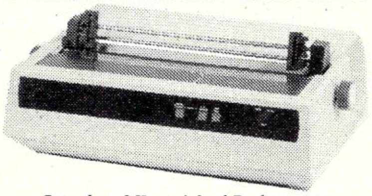
- 2 or 4 MHz operation
- Designed to IEEE proposed S-100 bus standards
- Supports IMSAI-type front panels
- Operates with either an 8080 or Z-80 based S-100 system providing processor-transparent refreshes with both
- Bank-select system allows system memory expansion
- Bank-select port's address is jumper selectable
- Any 16K block can be made bank-independent
- All 64K can be made bank-enabled on power-on and reset
- Fully buffered address and data lines
- Configuration as a 16K, 32K, or 48K board without the removal of RAMs
- Fail-safe refresh circuitry for extended Wait States
- Board configuration w/reliable easy-to-configure Berg jumpers
- Supports DMA
- Jumper-selectable Phantom input
- Uses Popular 4116 RAMs
- Assembled & tested
- All ICs in sockets
- Full Factory Warranty

SICCS20653 List Price $350.00

SALE $249.00 ea. - 4/$880.00

FUJITSU FROM SELLUM

LETTER QUALITY PRINTERS



Speed and Unmatched Performance

Interfaces: • RS232C and Current Loop • Centronics type parallel interface • IEEE/488 All are DIP switch selectable.
Personality Protocols: • NEC5510 • DIABLO 630 • QUME Sprint 9 • IBM Personal Computer • ATARI (Centronics 737).
Special Features: • Z80 CPU • 12K ROM • Standard 16K Buffer • Optional 48K Buffer • 50 - 19.2K baud • Micro-coded alarm differentiates error conditions with pulse combinations • Intelligent bi-directional printing with logic seeking • Complete word processing features • Self test • Auto reprint • Auto clear error • Proportional spacing • Supports Automatic justification • Complete Vector plotting routines • Sheet feeder mode allows easy interface to most mechanical sheet feeders • 6 month warrantee • Quiet - 60db • Front panel forms control • Universal power supply 115/220V 50/60 Hz.

BEST BUY! **80 Characters per Second! LETTER QUALITY! USING THE FUJITSU SP830** **WOW!**

Part No.	Description	List Price	Our Price
SISLMF86	Intelligent Printer	$3495.00	$2795.00
	OPTIONS		
SISLMF86VFT	Vertical forms tractor		$200.00
SISLMF8648K*	48K RAM buffer		$300.00

Call for pricing on sheet feed options
*Must be ordered with the printer

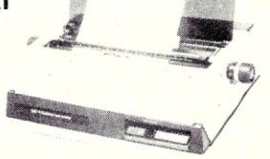
INTERFACE CABLES

SICPAM572	Centronics IBM/NEC Cable	$80.00
SISCDB251	RS232C Serial Cable	$60.00

Call for other cables not listed

SMITH-CORONA TP-1

LETTER QUALITY - ELECTRONIC TEXT PRINTER



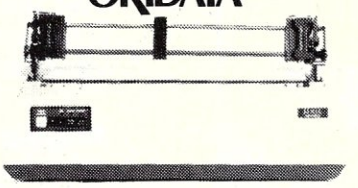
$649.00
SISCMTPS10 (Serial)
SISCMTP1P10 (Parallel)

• 12 CPS • 10 CPI • 6LPI • SERIAL OR PARALLEL INTERFACE • 50-19.2K Baud • Friction Feed • 88 Character

SISCM12625	TP1	Black Milar Ribbon	$3.50
SISCM12658	TP1	Black Fabric Ribbon	$7.50

FREE TRACTOR AND FREE OKIGRAPH!!

OKIDATA



COMPARE OKIDATA'S FEATURES TO ANY OTHER PRINTER! COMPARE PRIORITY ONE'S PRICES TO ANY — THEN DECIDE!

Don't be fooled by those who try to "soak" you for the options

- BI-DIRECTIONAL-120 CPS
- 9 x 9 Matrix (Alphanumeric)
- 6x9or12 Matrix for graphics
- 5,8.3, 10, 16 Characters Per Inch
- 80 CPL @ 10 CPI for 82A
- 132 CPL @ 100CPI for 83A & 84
- Both Parallel and 1200 baud Serial I/O (82A & 83A)
- Self Test
- Out of Paper Switch
- Friction or Optional Tractor Feed (Tractor Standard on 84)
- 10 different character sets

All three printers feature front panel switch selectable for lengths; 10 lengths from 3" to 14". Front or bottom aper loading for up to 4 part forms and tear bar. All of these features make OKIDATA the best value in low cost printers!

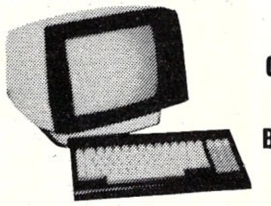
Part No.	Description	List Price	SALE
SIPDBOKI82ATG	OKIDAT82A w/tractor and graphics		$539.00
SIPDBOKI83ATG	OKIDAT83A w/tractor & graphics		$750.00
SIOKISER2KBF	9600 Baud 2K serial buffer/interface		$149.00
SIOKIDAT84AP	132 col. printer parallel interface with graphics and tractor (35 lbs.)		$ 1095.00
SIOKIDAT84AS	132 col. printer 9600 baud serial 2K interface, graphics and tractor (35 lbs)		$ 1250.00
SIMBSAPLWTWC*	Apple Parallel Interface w/Cable		$ 75.00

*Sold only with printer

COMPARE PRICES!!

ADDS

Applied Digital Data Systems Inc.



SALE!
Green Screen $525.00
Black & White $479.00

VIEWPOINT — ADDS

Detachable keyboard, RS232 interface and auxiliary port, 80 x 24 display, tiltable screen.

SIADDVWPR	ADDS 3A, P31 Green Phosphor	$699.00	$525.00
SIADDVWPRBW	P4 Black & White	$699.00	$479.00

(Shipping Weight 30 lbs.)

TeleVideo

WORLD'S BEST SELLING TERMINAL!

Extra Memory Pages! FREE!!

SIPDBTLV9252P* $749.00
*TeleVideo 925 w/free 2nd page memory kit, a $95.00 value!
SIPDBTLV9504P* $949.00
*TeleVideo 950 w/free 2nd, 3rd & 4th page memory kit, $285.00 value
SITLV910 Sh. Wt. 43 lbs. $599.00
BEST BUY!

SEND $1.00 TODAY FOR THE NEW, FULL COLOR SPRING 1982 ENGINEERING SELECTION GUIDE!

Tandon

Tandon TM-800 Thinline is exactly half the size of conventional 8" floppy disk drives.

ONE

Exactly one-half the height of any other model.
Proprietary, high-resolution, read-write heads patented by Tandon
D.C. only operation - no A.C. required
Industry standard interface.
Three millisecond track-to-track access time (9 lbs.)

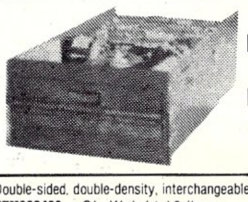
SITNDTM8481	Single Sided	$380.00	2 or more	$370.00
SITNDTM8482	Double Sided	$495.00	2 or more	$485.00
SITNDTM8M	Manual - not included with drive			$ 10.00

TANDON 5¼" DRIVES

SITNDTM1001	Single Sided, 250KB (5 lbs.)	$220.00* ea
	2 or More $200.00	
SITNDTM1002	Double Sided, 500KB	$295.00 ea.
	2 or More $270.00	
SITNDTM1003	Single Sided, 500KB	$295.00 ea.
	2 or More $270.00	
SITNDTM1004	Double Sided, 1000KB	$395.00 ea.
	2 or More $375.00	
SITNDTM5M	manual, not included with drive	$ 10.00

*As used in the IBM P.C.

MITSUBISHI ELECTRIC



Better Than QUME! Better Than SHUGART!

8", Double-sided, double-density, interchangeable with QUME & Shugart

SIMITM289463	Sh. Weight 16 lbs.	$495.00
SIMITM289463 M	Manual	$ 10.00

2 or More $475.00 each

Shugart 801R

Single sided, double density - most popular 8" drive

SISHU801R $394.90 ea. or 2 or more (16 lbs) for $389.90
SISHUSA801RM Manual for 801R drives $ 10.00

INTERNATIONAL INSTRUMENTATION, INC.

OUR BEST!!

UNIVERSAL DISK ENCLOSURES

• Accepts any combination of 8" drives (QUME/Shugart 801R type or ½ size Tandon type) • Also accepts hard disks • Positive pressure • Optional Disk environment monitor shows supply voltage and internal cabinet temperature • Internal power and data cables provided.

UNIVERSAL DRIVE CABINET complete with power supply, fan and filter, and all internal cables for attachment of two 8" floppy drives.

SIIIUDE004	Dual Drive Cabinet (Sh. Wt. 40 lbs)	$495.00	$450.00
SIIIUDE004	With purchase of two 8" Disk Drives		$399.00
SIIIUDE004EM	w/Environmental Monitor installed	$584.95	$535.00
SIIIUDE004EM	With purchase of two 8" Disk Drives		$485.00
SIIIUDE004AUG	Dual Drive cabinet with Augmentation power supply module to increase 5V supply from 6 amps to 9 amps for use with two hard disks or 4 Tandon drives. Also includes Disk Environment Monitor (45 lbs.)	$733.00	$650.00
SIIIUDERCK	19" Rack Mount kit UDE004		$ 89.95

THIN THREE DRIVE CABINET complete with power supply, and all internal cables for attachment of three thin 8" floppy drives (Tandon type). One AC power connector is also provided for use with full size drive.

SIIIUDETT	Three Drive Cabinet (Sh. Wt. 35 lbs)	$495.00	$450.00
SIIIUDETTTEM	with Environmental Monitor installed	$584.95	$535.00

PRIORITY ONE ELECTRONICS

VISA · master charge · NEDA · MEMBER MITA

9161 DEERING AVE • CHATSWORTH, CA 91311

ORDER TOLL FREE (800) 423-5922 - CA, AK, HI CALL (213) 709-5464

Terms: U.S. VISA, MC, BAC, Check, Money Order, U.S. Funds Only. CA residents add 6½% Sales Tax. MINIMUM PREPAID ORDER $15.00. Include MINIMUM SHIPPING & HANDLING of $3.00 for the first 3 lbs., plus 40¢ for each additional pound. Orders over 50 lbs. sent freight collect. Just in case, please include your phone number. Prices subject to change without notice. We will do our best to maintain prices through September, 1982. Credit Card orders will be charged appropriate freight. If you haven't received your Spring '82 Engineering Selection Guide, send $1.00 for your copy today! Sale prices are for prepaid orders only.

ated through strong data typing. Ennumeration types with explicitly declared subranges are provided for the use of strong typing.

Prolog

After observing that predicate logic can express anything that a typical programming language can express, Prolog, a language based on logic, was developed.

It is unusual in its declarative nature, which contrasts with the procedural style of most programming languages. In Prolog, a program consists of axioms, and theorems of deduction that the language may use in attempt to prove a goal.

For example, if Prolog can prove that a sentence is syntactically well formed, given the BNF of the language, the resultant proof is simply the parse of the sentence. This is exactly what the parser of any high-level language must do.

SETL

Proclaimed by its designers to be a "very" high-level language, SETL exploits the use of mathematical sets as the fundamental datatype. Though sets appeared in prior languages, SETL provides as primitives very high-level operations such as power set construction and the universal and existential quantifiers of first-order predicate calculus.

The idea behind SETL is to allow quite abstract specifications of programs that can actually be run, and then to allow parts to be rewritten for efficiency (though you have to do that yourself).

Ada

With the continuing effort of the Department of Defense and your tax dollars, the Ada Language is currently coming to fruition. Based on Pascal, the language includes no startling innovations. It is an excellent combination of concepts and ideas of high-level programming languages and it fulfills all the requirements that spurred its creation: software cost reduction, transportability, ease of maintenance, high reliability, readability, and efficient execution.

C and Forth

Since this issue of *Microsystems* contains reviews of C and Forth, let me briefly say why they are not discussed here, too.

Neither C nor Forth have brought with them any new ideas in high-level programming languages. This is not a shortcoming. In fact, they are

both what some people like to call MLLs (Medium Level Languages). MLLs are designed for working at the hardware level, yet they often provide powerful tools (control structure, data structure, scoping) that assemblers do not have.

Conclusion/Future Trends

Ada is certainly a fine product of state of the art work. While it does not present any totally original innovations, the selected ideas that it does embody, and the widespread use expected of it, suggest that Ada will have a large impact on future research in programming languages.

Ada does not encompass all the good ideas of past languages. In particular, APL, Lisp, Smalltalk, Snobol and SETL will continue to demonstrate the power and usefulness of languages that are based on a small core of consistent ideas. Also, Prolog exemplifies the move toward non-declarative programming that is expected in the future.

Another important area is the study of pushing high-level language concepts down into the hardware of the machine (e.g. VLSI). For example, the table idea of Snobol, a seemingly expensive operation, is realizable directly by using associative memory. Examination of microprocessors such as the Intel 8086 or Zilog Z8000 uncovers architectures specially modeled for handling activation records in block-structured languages such as Pascal or Algol.

Use of microcode is a quick way of creating suitable machines for specialized languages (i.e. the Symbolics Lisp Machine). Thus, innovation in high-level language can lead to innovation in hardware, giving us the benefit in the form of faster execution speed, increased reliability and higher overall capacity in our work.

In the future, language innovation will continue through research in the following areas: abstraction, verification, reliability, functional programming, database management, distributed systems and many ideas in artificial intelligence (plan generation, natural language, problem solving, etc.). It should not be unexpected to see specialized programming languages emerging from any one of these topics.

References

Ada Reference Manual, Defense Advanced Research Projects Agency, November 1980.

BAER, Jean-Loup, *Computer Systems Architecture*, Computer Science Press, Rockville MD, 1980.

High-Level Language *continued . . .*

BIRTWISTLE, Graham; Ole-Johan Dahl; Bjorn Myhrhaugi and Kristen Nygaard, *SIMULA Begin*, Auerbach, Philadelphia PA, 1973.

FELDMAN, Jerome, "Programming Languages," *Scientific American*, Vol. 241, No. 6, NY NY, December 1979.

GILMAN, Leonard, and Rose, Allen, *APL—An Interactive Approach*, John Wiley & Sons, Inc., NY NY, 1976.

GRISWOLD, R.E., Poage, J.F., and Polonsky, I.P., *The Snobol4 Programming Language*, Prentice-Hall, Inc., Englewood Cliffs NJ, 1971.

JENSEN, Kathleen and Wirth, Niklaus, *Pascal User Manual and Report*, Springer-Verlag, NY NY, 1974.

KOWALSKI, Robert, "Predicate Logic as Programming Language," *Information Processing* 74, North-Holland Publishing Company, 1974.

LEDGARD, Henry, *Ada—An Introduction*, Springer-Verlag, NY NY, 1981.

NELSON, Ted, *Computer Lib*, Hugo's Book Service, Chicago IL, 1974.

SAMMET, Jean E., "An Overview of High-Level Languages," *Advances in Computers*, Vol. 20, Academic Press, Inc., 1981.

WEISSMAN, Clark, *LISP 1.5 Primer*, Dickenson Publishing Company, Inc., Belmont CA, 1967.

WEXELBLAT, Richard L., *History of Programming Languages*, ACM SIGPLAN, Academic Press, NY NY, 1981.

WILLIAMS, Gregg, BYTE, Vol. 5, Num. 8, BYTE Publications, Peterborough NH, August 1980.

Xerox Learning Research Group, BYTE, Vol. 6, Num. 8, BYTE Publications, Peterborough NH, August 1981.

Don Libes recently received his M.S. in Computer Science from the University of Rochester and has a B.A. from Rutgers University in Mathematics. Don is interested in high-level programming environments, including problems of natural language and graphics interfaces. He is also trying hard to catch up with his father, Sol Libes.

Printer & Disk Drive Sale

Dot Matrix Printers

NEW EPSONS with GRAFTRAX-plus

Epson has improved and upgraded their best selling line of printers to include their new GRAFTRAX-plus graphics package. Features now include; 9 x 9 matrix, bi-directional/logic seeking, line spacing to n/216, programmable forms length and horizontal tabs, skip over perf, italics fonts, international symbols, superscript/subscript, normal, emphasized, double-strike, and double-emphasized print, underlining, line drawing graphics, 60/120 DPI bit image, software reset, adjustable right margin, and true back space.

MX-80 with GRAFTRAX-plus *80/132 column, 80 CPS, adjustable pin feed, parallel interface.*

PRM-28080 *MX-80 w/GRAFTRAX-plus* **$479.95**

MX-80FT with GRAFTRAX-plus *same as MX-80 with friction feed and pin feed.*

PRM-28082 *MX-80FT w/GRAFTRAX-plus* **$559.95**

MX-100 with GRAFTRAX-plus *132/232 column, correspondence quality, up to 15" paper, friction feed & adjustable pin feed, 18 x 18 dot matrix, 80 CPS.*

PRM-28100 *MX-100 w/GRAFTRAX-plus* **$754.95**

PRA-27084	*Serial interface*	**$54.95**
PRA-27088	*Serial intf & 2K buffer*	**$99.95**
PRA-27081	*Apple card*	**$39.95**
PRA-27082	*Apple cable*	**$19.95**
PRA-27086	*IEEE 488 card*	**$59.95**
PRA-27087	*TRS-80 cable*	**$24.95**
PRA-27097	*GRAFTRAX-plus MX-80, FT*	**$59.95**
PRA-27197	*GRAFTRAX-plus MX-100*	**$64.95**
PRA-27090	*MX-80, FT print head*	**$44.95**
PRA-27190	*MX-100 print head*	**$49.95**
PRA-27083	*MX-80 ribbon cart.*	**$13.95**
PRA-27101	*MX-100 ribbon only*	**$9.95**

BETTER THAN EPSON ! - Okidata

Microline 82A *80/132 column,* ***120 CPS****, 9 x 9 dot matrix, friction feed, pin feed, adjustable tractor feed (optional), handles 4 part forms up to 9.5" wide, rear & bottom feed, paper tear bar, 100% duty cycle/200,000,000 character print head, bi-directional/logic seeking,* ***both serial & parallel interfaces included****, front panel switch & program control of 10 different form lengths, uses inexpensive spool type ribbons, double width & condensed characters, true lower case descenders & graphics*

PRM-43082 *Friction & pin feed* **$499.95**

Microline 83A *132/232 column, 120 CPS, forms up to 15" wide, removable tractor, plus all the features of the 82A.*

PRM-43083 *with FREE tractor* **$744.95**

Microline 84 *132/232 column, Hi-speed 200 CPS, full dot graphics built in, plus all the features of the 83A.*

PRM-43084	*Centronics parallel*	**$1149.95**
PRM-43085	*Serial with 2K buffer*	**$1249.95**
PRA-27081	*Apple card*	**$39.95**
PRA-27082	*Apple cable*	**$19.95**
PRA-27087	*TRS-80 cable*	**$24.95**
PRA-43081	*2K hi speed serial card*	**$99.95**
PRA-43082	*Hi-res graphics ROMs 82A*	**$49.95**
PRA-43083	*Hi-graphics ROMs 83A*	**$49.95**
PRA-43088	*Tractor option for 82A*	**$49.95**
PRA-43080	*Extra ribbons pkg. of 2*	**$9.95**

8023 DOT MATRIX - NEC

100 CPS, proportional spacing, hi-resolution graphics, correspondence quality printing, bi-directional tractor & friction feed.

NEC-8023A	*8023 parallel*	**$499.95**
NEC-8023-01	*8023 ribbon*	**$11.95**

PRINTER PALS - F.M.J. Inc.

Desk top printer stand and continuous form paper holder.

PRA-99080	*for MX-80, MX-80FT, Oki 82A, NEC*	**$29.95**
PRA-99100	*for MX-100, Oki 83A & 84*	**$34.95**
PRA-99700	*for letter quality printers*	**$49.95**

Jade Bus Probe

THE BUS PROBE - Jade

Inexpensive S-100 Diagnostic Analyzer

Double-high S-100 board which displays all 96 IEEE S-100 signals without the expense of an ocilloscope or a front panel

TSX-200B	*Bare board*	**$59.95**
TSX-200K	*Kit*	**$119.95**
TSX-200A	*A&T*	**$149.95**

Dual Disk Sub-Systems

Disk Sub-Systems - Jade

Handsome metal cabinet with proportionally balanced air flow system, rugged dual drive power supply, power cable kit, power switch, line cord, fuse holder, cooling fan, never-mar rubber feet, all necessary hardware to mount 2-8" disk drives, power supply, and fan, does not include signal cable.

Dual 8" Sub-Assembly Cabinet

END-000420	*Bare cabinet*	**$59.95**
END-000421	*Cabinet kit*	**$225.00**
END-000431	*A & T*	**$359.95**

8" Sub-Systems - Single Sided, Double Density

END-000423	*Kit w/2 FD100-8Ds*	**$975.00**
END-000424	*A & T w/2 FD100-8Ds*	**$1175.00**
END-000433	*Kit w/2 SA-801Rs*	**$999.95**
END-000434	*A & T w/2 SA-801Rs*	**$1195.00**

8" Sub-Systems - Double Sided, Double Density

END-000426	*Kit w/2 DT-8s*	**$1224.95**
END-000427	*A & T w/2 DT-8s*	**$1424.95**
END-000436	*Kit w/2 SA-851Rs*	**$1274.95**
END-000437	*A & T w/2 SA-851Rs*	**$1474.95**

5¼" Disk Drives

Tandon TM100-1 *single-sided double-density 48 TPI*
MSM-551001 .... $248.95 ea 2 for **$219.95 ea**

Shugart SA400L *single-sided double-density 40 track*
MSM-104000 .... $234.95 ea 2 for **$224.95 ea**

Shugart SA455 *half-size double-sided 48 TPI*
MSM-104550 .... $349.95 ea 2 for **$329.95 ea**

Shugart SA465 *half-size doule-sided 96 TPI*
MSM-104650 .... $399.95 ea 2 for **$379.95 ea**

Tandon TM100-2 *double-sided double-density 48 TPI*
MSM-551002 .... $324.95 ea 2 for **$298.95 ea**

Shugart SA450 *double-sided double-density 35 track*
MSM-104500 .... $349.95 ea 2 for **$329.95 ea**

Tandon TM100-3 *single-sided double-density 96 TPI*
MSM-551003 .... $324.95 ea 2 for **$298.95 ea**

Tandon TM100-4 *double-sided double-density 96 TPI*
MSM-551004 .... $448.95 ea 2 for **$419.95 ea**

MPI B-51 *single-sided double-density 40 track*
MSM-155100 .... $234.95 ea 2 for **$224.95 ea**

MPI B-52 *double-sided double-density 40 track*
MSM-155200 .... $344.95 ea 2 for **$334.95 ea**

MPI B-91 *single-sided double-density 77 track*
MSM-155300 .... $369.95 ea 2 for **$359.95 ea**

MPI B-92 *double-sided double-density 77 track*
MSM-155400 .... $469.95 ea 2 for **$459.95 ea**

5¼" Cabinets with Power Supply

END-000216	*Single cab w/power supply*	**$69.95**
END-000226	*Dual cab w/power supply*	**$94.95**

8" Disk Drives

Shugart SA810 *half-size single-sided double-density*
MSF-108100 .... $424.95 ea 2 for **$394.95 ea**

Shugart SA860 *half-size double-sided double-density*
MSF-108600 .... $574.95 ea 2 for **$549.95 ea**

Shugart SA801R *single-sided double-density*
MSF-10801R .... $394.95 ea 2 for **$389.95 ea**

Shugart SA851R *double-sided double-density*
MSF-10851R .... $554.95 ea 2 for **$529.95 ea**

Tandon TM848-1 *single-sided double-den thin-line*
MSF-558481 .... $424.95 ea 2 for **$394.95 ea**

Tandon TM848-2 *double-sided double-den thin-line*
MSF-558482 .... $574.95 ea 2 for **$549.95 ea**

Qume DT-8 *double-sided double-density*
MSF-750080 .... $524.95 ea 2 for **$498.95 ea**

Mitsubishi M2894-63 *double-sided double-density*
MSF-289463 .... $549.95 ea 2 for **$524.95 ea**

Siemens FDD 100-8 *single-sided double-density*
MSF-201120 .... $384.95 ea 2 for **$349.95 ea**

Prices may be slightly higher at our retail locations. Please contact the store in your area for exact pricing.

Hard Disk Drives

12 MEGABYTE SUB-SYSTEM - DayStar

12 Megabyte Winchester sub-system for most CP/M based microcomputers including Apple II, S-100, IBM PC, NEC, Xerox, TRS-80 II, Heath H-89, and others. Sub-system includes interface, controller, cabinet, power supply, CP/M software patch, & cables - ready to just plug in and run. A 1 Megabyte back-up drive is also available.

MSH-531010	*S-100 sub-system*	**$2095.00**
MSH-531020	*Apple II sub-system*	**$2095.00**
MSH-531030	*NEC sub-system*	**$2095.00**
MSH-531040	*Xerox sub-system*	**$2095.00**
MSH-531050	*TRS-80 II sub-system*	**$2095.00**
MSH-531060	*IBM PC sub-system*	**$2095.00**
MSH-531099	*1 Megabyte back-up*	**$699.95**

12 MEGABYTE WINCHESTER - Tandon

5¼" hard disk drive for hi-speed mass storage.

MSH-550603 *Bare 5¼" drive* **$1094.95**

Letter Quality Printers

TP-1 LETTER QUALITY - SCM

10 CPS daisy wheel printer from Smith Corona.

PRD-45101	*Centronics parallel*	**$699.95**
PRD-45102	*Rs-232C serial*	**$749.95**

LETTER QUALITY PRINTER - Jade

Uses standard daisy wheels and ribbon cartridges, 16 CPS bi-directional printing, semi-automatic paper loader (single sheet or fan fold), 10/12/15 pitch, up to 16" paper, built-in noise suppression cover.

PRD-11001	*Centronics parallel*	**$959.95**
PRD-11002	*RS-232C serial model*	**$999.95**

STARWRITER F-10 - C. Itoh

New 40 CPS daisy wheel printer with full 15" carriage, uses standard Diablo print wheels and ribbons, both parallel and serial interfaces included.

PRD-22010 *Starwriter F-10* **$1495.95**

80 CPS LETTER QUALITY - Fujitsu

High speed daisy wheel printer with both RS-232C serial & Centronics parallel interfaces, emulates NEC 5510, Diablo 630, Qume, and will interface to the IBM Personal Computer, features include Z-80 CPU, 16K buffer (48K optional), bi-directional printing, & baud rates up to 19.2K

PRD-86100	*Fujitsu with 16K*	**$2895.00**
PRD-86200	*Fujitsu with 48K*	**$3195.00**
PRA-86000	*Adjustable tractor*	**$190.00**

Software

PLANNER CALC - Target Software

Spread sheet (what if ?) program designed with the user in mind, user oriented (simple english) commands allow you to quickly master this powerful software package, supplied on disk for most CP/M based systems.

SFA-12251052M	*Apple II 5¼" CP/M*	**$49.95**
SFC-12251050F	*SS SD 8" CP/M*	**$49.95**
SFC-12251053F	*Xerox 8" CP/M*	**$49.95**
SFC-12251053M	*Xerox 5¼" CP/M*	**$49.95**
SFX-12251051M	*NEC 5¼" CP/M*	**$49.95**

PAC MAN - Atari

Atari's best selling game for your 800 or 400 computer.

SFI-31254022E *Pac Man cartridge* **$37.95**

Place Orders Toll Free

Continental U.S.	Inside California
800-421-5500	800-262-1710

For Technical Inquires or Customer Service call:
213-973-7707

JADE
Computer Products

4901 W. Rosecrans, Hawthorne, Ca 90250

TERMS of SALE: Cash, checks, credit cards, or Purchase Orders from qualified firms and institutions. **Minimum Order $15.00.** California residents add 6% tax. Minimum shipping & handling charge $3.00. **Pricing & availibility subject to change**

CIRCLE 18 ON READER SERVICE CARD

Disk Drive for Apple $319.95

Diskettes

PREMIUM DISKETTES - Jade

We proudly put our name on these high quality diskettes -

5¼" Diskettes, Box of Ten

MMD-5110103	SS, SD, 01S	$29.00
MMD-5111003	SS, SD, 10S	$29.00
MMD-5111603	SS, SD, 16S	$29.00
MMD-5120103	SS, DD, 01S	$31.00
MMD-5121003	SS, DD, 010	$31.00
MMD-5121603	SS, DD, 16S	$31.00

8" Diskettes, Box of Ten

MMD-8110103	SS, SD, 01S	$31.00
MMD-8120103	SS, DD, 01S	$39.00
MMD-8220103	DS, DD, 01S	$48.00

BARGAIN DISKETTES

MMD-5110105	5¼" SS, DD, 01S	$19.95
MMD-5220105	5¼" DS, DD, 01S	$32.95
MMD-8110105	8" SS, SD, 01S	$24.95
MMD-8120105	8" SS, DD, 01S	$32.95
MMD-8220105	8" SS, DD, 01S	$35.95

Video Monitors

HI-RES 12" GREEN - Zenith

15 MHz bandwith 700 lines/inch, P31 green phosphor, switchable 40 or 80 columns, small, light-weight & portable.

VDM-201201	List price $189.95	$129.95

12" GREEN SCREEN - NEC

20 MHz bandwidth, P31 phosphor ultra-high resolution video monitor with audio.

VDM-561200	List price $289.95	$199.95

12" COLOR MONITOR - NEC

High resolution color monitor with audio.

VDC-651212	Color monitor	$389.95
NEC-1202D	RGB color monitor	$999.95

13" COLOR MONITORS - BMC

18 MHz RGB & composite video color monitors.

VDC-421320	13" RGB Color	$329.95
VDC-421310	13" Composite video	$299.95
VDX-420090	RGB card for Apple	$149.95

COLOR MONITORS - Amdek

Reasonably priced color video monitors.

VDC-80130	13" Color I	$379.95
VDC-801320	13" Color II	$894.95
IOV-2300A	DVM board for Apple	$199.95

AMBER or GREEN MONITORS - Jade

High resolution 18 MHz compact video monitors.

VDM-751210	12" Amber phosphor	$149.95
VDM-751220	12" Green phosphor	$139.95
VDM-750910	9" Amber phosphor	$149.95
VDM-750920	9" Green phosphor	$139.95

Modems

SIGNALMAN - Anchor

Direct-connect automatic answer/originate selection, 300 Baud full duplex, Bell 103, includes RS-232 cable

IOM-5600A	Signalman	$98.50

SMARTMODEM - Hayes

Sophisticated direct-connect auto-answer/auto-dial modem, touch-tone or pulse dialing, RS-232C interface, programmable

IOM-5400A	Smartmodem	$248.95
IOK-1500A	Hayes Chronograph	$218.95
IOM-2010A	Micromodem II	$328.95
IOM-1100A	Micromodem 100	$368.95

EPROM Erasers

ULTRA-VIOLET EPROM ERASERS

Inexpensive erasers for industry or home.

XME-3100A	Spectronics w/o timer	$69.50
XME-3101A	Spectronics with timer	$94.50
XME-3200A	Economy model	$39.95

Single User System

THREE BOARD SET - SD Systems

4 MHz Z-80A CPU, 64K RAM (optional 256K), serial I/O port, parallel I/O port, double density disk controller, CP/M 2.2 & manual set, system monitor, control & diagnostic software. Includes SBC-200, 64K ExpandoRAM II, Versafloppy II, & CP/M 2.2 - all boards are assembled & tested.

Board set with 64K of RAM	$1095.00
Board set with 256K of RAM	$1295.00

Apple II Accessories

16K RAM CARD - for Apple II

Expand your Apple to 64K, 1 year warranty

MEX-16700A	Save $125.00 !!!	$69.95

ADD-ON DISK DRIVE - for Apple II

Inexpensive direct replacement for Apple Disk II, works with Apple II controller as first or second drive.

MSM-123200	Add On Drive	$319.95
MSM-123100	Controller	$94.95

Z-80 CPU CARD - for Apple II

Two computers in one, Z-80 & 6502, more than doubles the power and potential of your Apple, includes Z-80 CPU card CP/M and complete manual set.

CPX-62800A	A & T with software	$249.95

8" DISK CONTROLLER - Vista

New from Vista Computer, single or double sided, single or double density, compatible with DOS 3.2/3.3, Pascal, & CP/M 2.2, Shugart & Qume compatible

IOD-2700A	A & T	$499.95

2 MEGABYTES for Apple II

Complete package includes: Two 8" double-density disk drives, Vista double-density 8" disk controller, cabinet, power supply, & cables, DOS 3.2/3.3, CP/M 2.2, & Pascal compatible.

1 MegaByte Package	Kit	$1495.00
1 MegaByte Package	A & T	$1695.00
2 MegaByte Package	Kit	$1795.00
2 MegaByte Package	A & T	$1995.95

APPLE-CAT - Novation

Software selectable 1200 or 300 baud, direct connect, auto-answer/auto-dial, auxiliary 3-wire RS232C serial port for printer.

IOM-5232A	Save $50.00!!!	$325.95

VISION 80 - Vista Computer

80 column x 24 line video card for Apple II, 128 ASCII characters, upper and lower case, 9 x 10 dot matrix with 3 dot descenders, standard data media terminal control codes, CP/M Pascal & Fortran compatible, 50/60 Hz

IOV-2400A	Vista Vision 80	$299.95

CPS MULTICARD - Mtn. Computer

Three cards in one! Real time clock/calendar, serial interface, & parallel interface - all on one card.

IOX-2300A	A & T	$179.95

HI-RES GRAPHICS CARD - Genie

Intelligent printer interface and control card allows full high resolution graphics and screen dumps.

IOP-2405A	Genie for Epson	$119.95
IOP-2410A	Genie for Okidata	$119.95
IOP-2415A	Genie for NEC/C. Itoh	$119.95

Power Strips

ISOBAR - GSC

Isolates & protects your valuable equipment from high voltage spikes & AC line noise, inductive isolated ground, 15 amp circuit breaker, U.L. listed

EME-115103	3 socket	$39.50
EME-115105	4 socket	$49.50
EME-115100	8 socket	$54.50
EME-115110	9 socket rackmount	$74.50

Single Board Computer

SUPERQUAD - Adv. Micro Digital

Single board, standard size S-100 computer system, 4 MHz Z-80A, single or double density disk controller for 5¼" or 8" drives, 64K RAM, extended addressing, up to 4K of EPROM, 2 serial & 2 parallel I/O ports, real time interrupt clock, CP/M compatible.

CPC-30800A	A & T	$724.95
IOX-4232A	Serial I/O adapter	$29.95

Z-80 STARTER KIT - SD Systems

Complete Z-80 microcomputer with RAM, ROM, I/O, keyboard, display, kludge area, manual, & workbook.

CPS-30100K	Kit with workbook	$299.95
CPS-30100A	A & T with workbook	$469.95

AIM-65 - Rockwell International

Complete 6502 microcomputer with alphanumeric display, printer, keyboard, & instruction manual.

CPK-50165A	1K AIM-65	$424.95
CPK-50465A	4K AIM-65	$474.95
SFK-74600008E	8K Basic ROM	$64.95
SFK-64600004E	4K assembler ROM	$43.95
SFK-74600020E	PL/65 ROM	$84.95
SFK-74600010E	Forth ROM	$64.95
SFK-74600030E	Instant Pascal	$99.95
PSX-030A	Power supply	$64.95
ENX-000002	Enclosure	$54.95

SPECIAL PACKAGE

4K AIM-65, 8K Basic, power supply, & enclosure

Special Package Price	$649.95

S-100 EPROM Boards

PROM-100 - SD Systems

2708, 2716, 2732 EPROM programmer with software.

MEM-99520K	Kit with software	$189.95
MEM-99520A	A & T with software	$249.95

PB-1 - SSM Microcomputer

2708, 2716 EPROM board with on-board programmer.

MEM-99510K	Kit with manual	$154.95
MEM-99510A	A & T with manual	$219.95

EPROM BOARD - Jade

16K or 32K uses 2708 or 2716 EPROMs, 1K boundary.

MEM-16230K	Kit w/o EPROMs	$79.95
MEM-16230A	A & T w/o EPROMs	$119.95

S-100 Video Boards

SPECTRUM COLOR - CompuPro

Full-function color graphics board, up to 8 colors, 256 x 192 graphics, parallel I/O port, 8K RAM.

IOV-1870A	A & T	$348.95
IOV-1870C	CSC	$398.95

MICROANGELO - Scion

Ultra-high-resolution 512 x 480, 256 color or black & white S-100 video board

IOV-1500A	A & T	$999.95

S-100 MotherBoards

ISO-BUS - Jade

Silent, simple, and on sale - a better motherboard

6 Slot (5¼" x 8⅝")

MBS-061B	Bare board	$19.95
MBS-061K	Kit	$39.95
MBS-061A	A & T	$59.95

12 Slot (9¾" x 8⅝")

MBS-121B	Bare board	$29.95
MBS-121K	Kit	$69.95
MBS-121A	A & T	$99.95

18 Slot (14½" x 8⅝")

MBS-181B	Bare board	$49.95
MBS-181K	Kit	$99.95
MBS-181A	A & T	$139.95

ACTIVE TERMINATOR - CompuPro

A true mother's helper.

TSX-100A	A & T	$59.45

Prices may be slightly higher at our retail locations. Please call the store nearest you for local price and availability.

CompuPro Boards on Sale

S-100 CPU Boards

8086/8087 - CompuPro
16 bit, 8 or 10 MHz 8086 CPU with provisions for 8087 & 80130.

CPU-70520A	*8 MHz 8086 A & T*	**$624.95**
CPU-70520C	*8 MHz 8086 CSC*	**$764.95**
CPU-70530A	*with 8087 A & T*	**$1224.95**
CPU-70530C	*with 8087 CSC*	**$1455.95**

8085/8086 - CompuPro
Both 8 & 16 bit CPUs, standard 8 bit S-100 bus, up to 8 MHz, accesses 16 Megabytes of memory.

CPU-20510A	*6 MHz A & T*	**$398.95**
CPU-20510C	*6/8 MHz CSC*	**$497.95**

CPU-Z - CompuPro
2/4 MHz Z80A CPU, 24 bit addressing.

CPU-30500A	*2/4 MHz A & T*	**$279.95**
CPU-30500C	*3/6 MHz CSC*	**$374.95**

SBC-200 - SD Systems
4 MHz Z-80A CPU with serial & parallel I/O, 1K RAM, 8K ROM space, monitor PROM included.

CPC-30200A	*A & T*	**$399.95**

THE BIG Z - Jade
2 or 4 MHz switchable Z-80 CPU board with serial I/O, accomodates 2708, 2716, or 2732 EPROM, baud rates from 75 to 9600.

CPU-30201B	*Bare board w/manual*	**$35.00**
CPU-30201K	*Kit with manual*	**$149.95**
CPU-30210A	*A & T with manual*	**$199.95**

CB-2 - SSM Microcomputer
2 or 4 MHz Z-80 CPU board with provision for up to 8K of ROM or 4K of RAM on board, extended addressing, IEEE S-100, front panel compatible.

CPU-30300K	*Kit with manual*	**$229.95**
CPU-30300A	*A & T with manual*	**$274.95**

2810 Z-80 CPU - C.C.S.
2 or 4 MHZ Z-80 CPU with serial I/O port & on-board monitor PROM, front panel compatible.

CPU-30400A	*A & T with PROM*	**$289.95**

2820 Z-80 DMA CPU - C.C.S.
4 MHz Z-80 CPU board with 2 serial I/O ports & Centronics parallel I/O port, separate data & status ports, DMA daisy chain compatible.

CPU-30420A	*A & T with manual*	**$569.95**

S-100 Disk Controllers

DISK 1 - CompuPro
8" or 5¼" DMA disk controller, single or double density, single or double sided, 10 MHz.

IOD-1810A	*A & T*	**$449.95**
IOD-1810C	*CSC*	**$554.95**
SFC-52506580F	*8" CP/M 2.2 for Z-80*	**$174.95**
SFC-52506586F	*8" CP/M 2.2 for 8086*	**$299.95**
SFO-54158000F	*Oasis single user*	**$499.95**
SFO-54158002F	*Oasis multi-user*	**$849.95**

VERSAFLOPPY II - SD Systems
Double density disk controller for any combination of 5¼" and 8" single or double sided, analog phase-locked loop data separator, vectored interrupts, CP/M 2.2 & Oasis compatible, control/diagnostic software PROM included.

IOD-1160A	*A & T with PROM*	**$359.95**
SFC-55009047F	*CP/M 2.2 with VF II*	**$99.95**

2242 DISK CONTROLLER - C.C.S.
5¼" or 8" double density disk controller with on-board boot loader ROM, free CP/M 2.2 & manual set.

IOD-1300A	*A & T with CP/M 2.2*	**$399.95**

DOUBLE D - Jade
High reliablity double density disk controller with on-board Z-80A, auxiliary printer port, IEEE S-100, can function in multi-user interrupt driven bus.

IOD-1200B	*Bare board & hdwr man*	**$59.95**
IOD-1200K	*Kit w/hdwr & sftwr man*	**$299.95**
IOD-1200A	*A & T w/hdwr & sftwr man*	**$359.95**
SFC-59002001F	*CP/M 2.2 with Double D*	**$99.95**

S-100 Memory Boards

256K RAMDISK - SD Systems
ExpandoRAM III expandable from 64K to 256K using 64K x 1 RAM chips, compatible with CP/M, MP/M, Oasis, Cromemco, & most other Z-80 based systems, functions as ultra-high speed disk drive when used with optional RAMDISK software.

MEM-65064A	*64K A & T*	**$474.95**
MEM-65128A	*128K A & T*	**$574.95**
MEM-65192A	*192K A & T*	**$674.95**
MEM-65256A	*256K A & T*	**$774.95**
SFC-55009000F	*RAMDISK sftwr CP/M 2.2*	**$44.95**
SFC-55009000F	*RAMDISK with EXRAM III*	**$24.95**

128K RAM 21 - CompuPro
128K x 8 bit or 64K x 16 bit static RAM board, 12 MHz, 24 bit addressing.

MEM-12810A	*A & T*	**$1609.95**
MEM-12810C	*CSC*	**$1794.95**

64K RAM 17 - CompuPro
64K CMOS static RAM board, 10 MHz, low power less than 4 watts, DMA compatible, 24 bit addressing.

MEM-64180A	*64K A & T*	**$549.95**
MEM-64180C	*64K CSC*	**$698.95**

64K RAM 16 - CompuPro
32K x 16 bit or 64K x 8 bit low power static RAM board, 10 MHz, 24 bit addressing.

MEM-32180A	*RAM 16 A & T*	**$598.95**
MEM-32180C	*RAM 16 CSC*	**$698.95**

64K STATIC RAM - SSM
IEEE 696/S-100 standard, up to 6MHz/8 Bit, 12MHz/16 Bit, 24 Bit extended addressing, disable-able in 2K increments

MEM-64300A	*A & T*	**$499.95**

64K STATIC RAM - Mem Merchant
64K static S-100 RAM card, 4 to 16K banks up to 8 MHz.

MEM-64400A	*64K A & T*	**$499.95**

2065 64K RAM - C.C.S.
4 MHz bank port/bank byte selectable, extended addressing, 16K bank selectable, front panel compatible.

MEM-64565A	*64K A & T*	**$349.95**

2066 64K RAM - C.C.S.
64K RAM board with bank and block select switching functions for Cromemeco Cromix & Alpha Micro.

MEM-64566A	*64K A & T*	**$424.95**

64K EXPANDORAM II - SD Systems
Expandable RAM board from 16K to 64K using 4116 RAM chips.

MEM-16630A	*16K A & T*	**$344.95**
MEM-32631A	*32K A & T*	**$364.95**
MEM-48632A	*48K A & T*	**$384.95**
MEM-64633A	*64K A & T*	**$399.95**

MEMORY BANK - Jade
4 MHz S-100 bank selectable expandable to 64K.

MEM-99730B	*Bare board w/manual*	**$49.95**
MEM-99730K	*Kit with no RAM*	**$179.95**
MEM-32731K	*32K kit*	**$199.95**
MEM-64733K	*64K kit*	**$249.95**
Assembled & Tested		**add $50.00**

32K RAM 20 - CompuPro
32K static RAM, up to 10 MHz, disable-able in 4K banks, bank select or 24 bit addressing.

MEM-16180A	*16K A & T*	**$259.95**
MEM-16180C	*16K CSC*	**$324.95**
MEM-24180A	*24K A & T*	**$324.95**
MEM-24180C	*24K CSC*	**$384.95**
MEM-32185A	*32K A & T*	**$384.95**
MEM-32185C	*32K CSC*	**$449.95**

16K STATIC RAM - Mem Merchant
4MHz lo-power static RAM board, IEEE S-100, bank selectable, addressable in 4K blocks, disable-able in 1K segments extended addressing.

MEM-16171A	*16K A & T*	**$149.95**

S-100 I/O Boards

SYSTEM SUPPORT 1 - CompuPro
Real time clock, three 16 bit interval timers, dual interrupt controllers(15 levels), up to 4K EPROM/RAM, RS-232C serial channel, provision for 9511A/9512 math chip.

IOX-1850A	*SS1 A & T*	**$359.95**
IOX-1850C	*SS1 CSC*	**$459.95**
IOX-1855A	*with 9511 A & T*	**$554.95**
IOX-1855C	*with 9511 CSC*	**$654.95**
IOX-1860A	*with 9512 A & T*	**$554.95**
IOX-1860C	*with 9512 CSC*	**$654.95**

INTERFACER 1 - CompuPro
2 serial I/O ports 50-19.2K baud.

IOI-1810A	*A & T*	**$218.95**
IOI-1810C	*CSC*	**$288.95**

INTERFACER 2 - CompuPro
3 parallel, 1 serial, & interrupt timer.

IOI-1820A	*A & T*	**$218.95**
IOI-1820C	*CSC*	**$288.95**

INTERFACER 3 - CompuPro
5 or 8 channel serial I/O board for interrupt driven multi-user systems up to 250K baud.

IOI-1835A	*5 port A & T*	**$558.95**
IOI-1835C	*5 port CSC*	**$628.95**
IOI-1838A	*8 port A & T*	**$628.95**
IOI-1838C	*8 port CSC*	**$749.95**

INTERFACER 4 - CompuPro
3 serial, 1 parallel, 1 Centronics parallel.

IOI-1840A	*A & T*	**$314.95**
IOI-1840C	*CSC*	**$414.95**

MPX - CompuPro
Multi-user I/O multiplexer & interrupt controller with on-board 8085A-2 CPU & 4K or 16K of RAM.

IOI-1875A	*4K MPX A & T*	**$444.95**
IOI-1875C	*4K MPX CSC*	**$534.95**
IOI-1880A	*16K MPX A & T*	**$584.95**
IOI-1880C	*16K MPX CSC*	**$674.95**

I/O-8 - SSM Microcomputer
Eight software programmable serial I/O ports, 110 -19.2K Baud, ideal for multi-user systems

IOI-1018A	*A & T*	**$469.95**

I/O-5 - SSM Microcomputer
Two serial & 3 parallel I/O ports, 110-19.2K Baud

IOI-1015A	*A & T*	**$289.95**

MPC-4 - SD Systems
Intelligent 4-port serial I/O card, on-board Z-80A, 2K RAM, 4K PROM area, on-board firmware, fully buffered, vectored interrupts, four CTC channels, add to SD Board set for powerful multi-user system

IOI-1504A	*A & T w/software*	**$495.00**

I/O-4 - SSM Microcomputer
2 serial I/O ports plus 2 parallel I/O ports.

IOI-1010B	*Bare board w/manual*	**$35.00**
IOI-1010K	*Kit with manual*	**$179.95**
IOI-1010A	*A & T with manual*	**$249.95**

2830 6 PORT SERIAL - C.C.S.
Six asynchronous RS-232C serial I/O ports with programmable baud rates.

IOI-1040A	*A & T with manual*	**$529.95**

2710 4 PORT SERIAL - C.C.S.
Four RS-232C serial I/O ports with full handshaking.

IOI-1060A	*A & T with manual*	**$319.95**

2719 2 SER & 2 PAR - C.C.S.
Two RS-232C serial I/O ports plus two 8 bit parallel I/O ports.

IOI-1080A	*A & T with manual*	**$349.95**

CIRCLE 18 ON READER SERVICE CARD

Prices may be slightly higher at our retail locations. Please call the store nearest you for local price and availability.

An Ada Tutorial, Part I

by Mark M. Zeiger

In 1974, faced with increasing software costs and programs written in a multitude of different languages, the Department of Defense (DoD) decided to adopt an "official" language. Finding none of the "Algol-like" languages acceptable, they commissioned a number of different groups to submit specifications for a language. After a process of integrating and modifying the suggestions, they came up with a language they chose to call "Ada" in honor of Augusta Ada Byron, the daughter of Lord Byron and assistant to Charles Babbage (the inventor of the Analytic Machine—the first computer). Because Ada will be the language that all programmers will have to use when programming for the Department of Defense, it is undoubtedly going to become one of the important languages of the 1980's.

Ada is a big language and has many features not found in other popular languages. It is designed for easy readability, for strong typing and many data types, and for portability. Ideas for the language were accepted from a huge cross-section of programmers from all over the world to make sure that all modern language features were at least considered. Because Ada is so big, the chances of seeing a full implementation of the language on a microcomputer (or at least an eight-bit computer) are slim. However, a few software houses have already written modified versions of Ada (R&R Software has published Janus and Supersoft has written Ada); therefore implementations of Ada that run under CP/M are already realized. It should be noted, however, that the DoD will not accept any implementations of Ada that deviate from the published standard (*The Ada Language Reference Manual, MIL-STD 1815*), whether they be supersets or subsets. In fact, variations may not even be given the name "Ada." The DoD is really determined this time to have only one language.

Since we are going to be seeing bigger and better implementations of Ada on micros, this article is intended to give the reader some information on how to write in Ada. Just a casual glance at Ada will tell you that this is a Pascal-like language. In fact, with just a few notes on Ada syntax on hand, it is a very simple matter to take a complicated Pascal program and rewrite it in Ada. In this article, I will describe some of the features of the language by comparing it to Pascal. Then some of the more advanced features of Ada will be described. Since space is limited, I will not be able to discuss the whole language, so I will talk about the characteristics I personally found interesting. At the end of the article, I have listed a few references which may be read if you wish to learn more about Ada.

There are a few conventions I will use in this text. Any underlined word is an Ada (or Pascal) reserved word. Comments in Ada are indicated by using two dashes (--) followed by the comment. The comment may continue to the end of the line only. This is mentioned now because comments are used in program segments to describe some of the features of Ada. I will also use "squiggly" brackets to indicate Pascal comments, in accordance with the Pascal convention.

Program Structure

In its basic form, the structure of an Ada program is similar to Pascal. The overall structure of programs written in each of the languages would be:

```
program PASCAL_PROG;                      package body ADA_PROG is

  const PI = 3.1459;                        PI constant := 3.14159;
  type DAY = (MON, TUE, WED);               type DAY is (MON, TUE, WED);
  var I : integer;                          I : integer;
  WD   : DAY;                             WD: DAY;

procedure PROC1 (X : integer);            procedure PROC1 (X : integer) is
    {procedure declarations};               -- procedure declarations;
  begin                                     begin
    {procedure statements}                  -- procedure statements;
  end;                                      end PROC1;

function F1(Y : integer) : real;          function F1(Y : integer)
    {function declarations};                      return float is
  begin                                       --function declarations;
    {function statements}                   begin
  end;                                        --function statements;
                                            end F1;
  begin
    {program body}                          begin

  end.                                        --program body;

                                            end ADA_PROG;
```

As Pascal programmers know, each procedure may take on the structure of a program with local const, type, and var declarations and with its own

Mark M. Zeiger, 198–01B 67th Ave., Flushing, NY 11365.

procedures and functions. The same is true in Ada. However, the above structure did not show some of Ada's flexibility. In Pascal, const, type, and var declarations must occur in the order listed and in groups. This is not true in Ada. At the beginning of the Ada program we could have written:

```
integer;              --a variable declaration
constant PI := 3.14159;
type DAY is (MON, TUE, WED);
WD : DAY;             --another variable declaration
```

Naturally the definition of type DAY must precede the declaration of any variable of that type, but this is the only restriction on order. In Pascal, all variables which are used must be declared, while in Ada there are a few exceptions to this rule as we will see later.

Program Statements

Even though we haven't scratched the surface in describing variables and type declarations used in Ada, we will use the small amount already described in discussing executable statements.

Assignment Statements

The symbol := is used in Pascal and Ada in an identical manner. Ada does have an advantage over Pascal in that initial values may be assigned to variables at the time of declaration. For instance, the statement

```
DELTA : integer := 20;
```

declares DELTA to be an integer and assigns it an initial value of twenty. Furthermore, if this declaration occurs in a procedure, function, or block (blocks will be described later), then the variable will be set to that initial value each time the subroutine or block is invoked. Much nicer than initializing single variables is the ability to initialize composite types such as arrays and records. Compare the Ada declaration

```
A : array (1..3,1..3) of integer := ( (1,2,3)
                                       (4,5,6)
                                       (7,8,9)  );
```

to the inelegant Pascal statements:

```
A : array [1..3,1..3] of integer;
A[1,1] := 1; A[1,2] := 2; A[1,3] := 3;
A[2,1] := 4; A[2,2] := 5; A[2.3] := 6;
A[3,1] := 7; A[3,2] := 8; A[3,3] := 9;
```

The assignment statement often requires the use of arithmetic operators. Except as noted, operands may not be of different types. You may not add an integer and real number, even if the result is going to be assigned to a real variable. Later in the article we will see how to perform operations on different types.

The operations available in Ada are:

+	addition
−	subtraction
*	multiplication
/	division—Integer division truncates.
*Ø	exponentiation—second operator must be an integer. If the first operator is real, then the second may be an integer
rem	division remainder
mod	modulo remainder

The rem and mod operators are different. The result of the rem operation always has the sign of the first operand, so that 12 rem 5 and 12 rem −5 are both equal to 2, while −12 rem 5 and −12 rem −5 are equal to −2. With the mod operator, if we assume that the second operator is N, then the result of the mod operation is in the range O..N−1 if N is positive and in the range O...N+1 if N is negative.

As you've noticed, the semicolon is used in both Ada and Pascal, but the use is different in each language. In Pascal the semicolon is used as a statement separator, and it's safe to say that its use has befuddled many a beginning Pascal programmer. The semicolon is used in Ada as a statement terminator. Therefore one does not usually have a problem deciding when a semicolon is necessary or not. In Pascal the semicolon especially causes problems with if-then-else statements and near the end of blocks where many statements are ending. This problem does not occur in Ada. However, there is a logic to the use of the Pascal semicolon

Ada is a big language designed for easy readability, strong typing and portability. The chances of seeing a full implementation on a microcomputer are slim.

and one can get accustomed to it in a short while.

Control Statements

Ada and Pascal have essentially the same types of control structure and statements. They are the if-then-else statement, the case statement, and commands which accomplish loops.

If-then-else

Unlike Pascal, Ada is capable of executing multiple statements as a result of an if statement. Now you're probably saying that Pascal may also execute many statements if the boolean expression in the if clause evaluates to "true," but the actual syntax of Pascal is:

```
if <boolean expression> then <statement>
            {else <statement>};
```

Of course in Pascal a statement may be a group of statements in a begin-end block, so in reality the Pascal if-then is similar to Ada's. We will see, however, that nested conditionals in Ada are much easier to work with.

The actual format of Ada's if-then statement is:

```
if( <boolean expression>then
      statement 1;
      statement 2;
            .
            .
end if;
```

or

```
if <boolean expression>then
     statement 1;
     statement 2;
           .
           .
       statement—n;
else
       statement
           .
           .
end if;
```

The if-then statement also has an additional elseif clause.

This is used as follows:

```
if x = a then
   -- one or more statements;
elseif x = b then
   -- one or more statements;
elseif x = c then
   -- one or more statements;
end if;
```

It's pretty obvious that if x = a then the first group of statements will be executed, if x = b then the second group will be performed, and if x = c then control will pass to the third group. If x is not equal to a, b, or c then the program will execute the group of statements in the else clause. The else clause is of course optional or may even be present and contain the null statement. The null statement is Ada's "NOP". Notice that if the elseif command was not available, similar results could still be accomplished (as one would have to do in Pascal) by using the separate else and if commands. The code would be:

```
if x = a then                      -- start block 1
   -- statements;
else
   if x = b then                   -- start block 2
      -- statements;
   else
      if x = c then                --start block 3
         -- statements;
   else
            -- statements;
         end if;                   -- end block 3
    end if;                        -- end block 2
end if;                            -- end block 1
```

Obviously the first structure is much easier to understand and the programmer is less likely to match up the wrong end or else with the corresponding if. This goes along with Ada's philosophy of making programs readable, writeable, and error free.

Loops

Naturally Ada has provisions for loops. How could any high level language not have them? And Ada does a very nice job in this department. The basic

Nested conditionals are much easier to work with in Ada than in Pascal.

NEW! TPM* for TRS-80 Model II
NEW! System/6 Package
Computer Design Labs

Z80* Disk Software

We have acquired the rights to all TDL software (& hardware). TDL software has long had the reputation of being the best in the industry. Computer Design Labs will continue to maintain, evolve and add to this superior line of quality software.
— Carl Galletti and Roger Amidon, owners.

Software with Manual/Manual Alone

All of the software below is available on any of the following media for operation with a Z80 CPU using the CP/M* or similar type disk operating system (such as our own TPM*).

for TRS-80* CP/M (Model I or II)
for 8" CP/M (soft sectored single density)
for 5¼" CP/M (soft sectored single density)
for 5¼" North Star CP/M (single density)
for 5¼" North Star CP/M (double density)

BASIC I
A powerful and fast Z80 Basic interpreter with EDIT, RENUMBER, TRACE, PRINT USING, assembly language subroutine CALL, LOADGO for "chaining", COPY to move text, EXCHANGE, KILL, LINE INPUT, error intercept, sequential file handling in both ASCII and binary formats, and much, much more. It runs in a little over 12 K. An excellent choice for games since the precision was limited to 7 digits in order to make it one of the fastest around. $49.95/$15.

BASIC II
Basic I but with 12 digit precision to make its power available to the business world with only a slight sacrifice in speed. Still runs faster than most other Basics (even those with much less precision). $99.95/$15.

BUSINESS BASIC
The most powerful Basic for business applications. It adds to Basic II with random or sequential disk files in either fixed or variable record lengths, simultaneous access to multiple disk files, PRIVACY command to prohibit user access to source code, global editing, added math functions, and disk file maintenance capability without leaving Basic (list, rename, or delete). $179.95/$25.

ZEDIT
A character oriented text editor with 26 commands and "macro" capability for stringing multiple commands together. Included are a complete array of character move, add, delete, and display function. $49.95./$15.

ZTEL
Z80 Text Editing Language - Not just a text editor. Actually a language which allows you to edit text and also write, save, and recall programs which manipulate text. Commands include conditional branching, subroutine calls, iteration, block move, expression evaluation, and much more. Contains 36 value registers and 10 text registers. Be creative! Manipulate text with commands you write using Ztel. $79.95/$25.

TOP
A Z80 Text Output Processor which will do text formatting for manuals, documents, and other word processing jobs. Works with any text editor. Does justification, page numbering and headings, spacing, centering, and much more! $79.95/$25.

MACRO I
A macro assembler which will generate relocateable or absolute code for the 8080 or Z80 using standard Intel mnemonics plus TDL/Z80 extensions. Functions include 14 conditionals, 16 listing controls, 54 pseudo-ops, 11 arithmetic/logical operations, local and global symbols, chaining files, linking capability with optional linker, and recursive/reiterative macros. This assembler is so powerful you'll think it is doing all the work for you. It actually makes assembly language programming much less of an effort and more creative. $79.95/$20.

MACRO II
Expands upon Macro I's linking capability (which is useful but somewhat limited) thereby being able to take full advantage of the optional Linker. Also a time and date function has been added and the listing capability improved. $99.95/$25.

LINKER
How many times have you written the same subroutine in each new program? Top notch professional programmers compile a library of these subroutines and use a Linker to tie them together at assembly time. Development time is thus drastically reduced and becomes comparable to writing in a high level language but with all the speed of assembly language. So, get the new CDL Linker and start writing programs in a fraction of the time it took before. Linker is compatible with Macro I & II as well as TDL/Xitan assemblers version 2.0 or later. $79.95/$20.

DEBUG I
Many programmers give up on writing in assembly language even though they know their programs would be faster and more powerful. To them assembly language seems difficult to understand and follow, as well as being a nightmare to debug. Well, not with proper tools like Debug I. With Debug I you can easily follow the flow of any Z80 or 8080 program. Trace the program one step at a time or 10 steps or whatever you like. At each step you will be able to see the instruction executed and what it did. If desired, modifications can then be made before continuing. It's all under your control. You can even skip displaying a subroutine call and up to seven breakpoints can be set during execution. Use of Debug I can pay for itself many times over by saving you valuable debugging time. $79.95/$20.

DEBUG II
This is an expanded debugger which has all of the features of Debug I plus many more. You can "trap" (i.e. trace a program until a set of register, flag, and/or memory conditions occur). Also, instructions may be entered and executed immediately. This makes it easy to learn new instructions by examining registers/memory before and after. And a RADIX function allows changing between ASCII, binary, decimal, hex, octal, signed decimal, or split octal. All these features and more add up to give you a very powerful development tool. Both Debug I and II must run on a Z80 but will debug both Z80 and 8080 code. $99.95/$20.

ZAPPLE
A Z80 executive and debug monitor. Capable of search, ASCII put and display, read and write to I/O ports, hex math, breakpoint, execute, move, fill, display, read and write in Intel or binary format tape, and more! on disk $34.95/$15.

APPLE
8080 version of Zapple $34.95/$15.

NEW! TPM now available for TRS-80 Model II!

TPM*
A NEW Z80 disk operation system! This is not CP/M*. It's better! You can still run any program which runs with CP/M* but unlike CP/M* this operating system was written specifically for the Z80* and takes full advantage of its extra powerful instruction set. In other words its not warmed over 8080 code! Available for TRS-80* (Model I or II). Tarbell, Xitan DDDC, SD Sales "VERSA-FLOPPY", North Star (SD&DD), and Digital (Micro) Systems. $79.95/$25.

SYSTEM MONITOR BOARD (SMB II)
A complete I/O board for S-100 systems. 2 serial ports, 2 parallel ports, 1200/2400 baud cassette tape interface, sockets for 2K of RAM, 3-2708/2716 EPROM's or ROM, jump on reset circuitry. Bare board $49.95/$20.

ROM FOR SMB II
2KX8 masked ROM of Zapple monitor. Includes source listing $34.95/$15.

PAYROLL (source code only)
The Osborne package. Requires C Basic 2.
5" disks $124.95 (manual not included)
8" disks $ 99.95 (manual not included)
Manual $20.00

ACCOUNTS PAYABLE/RECEIVABLE (source code only)
By Osborne, Requires C Basic 2
5" disks $124.95 (manual not included)
8" $99.95 (manual not included)
Manual $20.00

GENERAL LEDGER (source code only)
By Osborne. Requires C Basic 2
5" disks $99.95 (manual not included)
8" disks $99.95 (manual not included)
Manual $20.00

C BASIC 2
Required for Osborne software. $99.95/$20.

SYSTEM/6
TPM with utilities, Basic I interpreter, Basic E compiler, Macro I assembler, Debug I debugger, and ZEDIT text editor.
Above purchased separately costs $339.75
Special introductory offer: Only $179.75 with coupon!!

$160.

This Coupon Is Worth One Hundred And Sixty Dollars Toward The Full Price Of The SYSTEM/6 Package
System/6 with this coupon is only $179.95.
This is a limited time offer.

$160.00

ORDERING INFORMATION
Visa, Master Charge and C.O.D. O.K. To order call or write with the following information.

1. Name of Product (e.g. Macro I)
2. Media (e.g. 8" CP/M)
3. Price and method of payment (e.g. C.O.D.) include credit card info. if applicable.
4. Name, Address and Phone number.
5. For TPM orders only: Indicate if for TRS 80, Tarbell, Xitan DDDC, SD Sales (5¼" or 8"). ICOM (5¼" or 8"), North Star (single or double density) or Digital (Micro) Systems.
6. N.J. residents add 5% sales tax.

Manual cost applicable against price of subsequent software purchase in any item except for the Osborne software.

For information and tech queries call
609-599-2146

These are NEW Toll Free numbers
For phone orders ONLY call toll free
1-800-458-3491 Ext. 15
In PA only 1-800-252-3567

OEMS
Many CDL products are available for licensing to OEMs. Write to Carl Galletti with your requirements.

* Z80 is a trademark of Zilog
* TRS-80 is a trademark for Radio Shack
* TPM is a trademark of Computer Design Labs. It is not CP/M*
* CP/M is a trademark of Digital Research
Prices and specifications subject to change without notice.

DEALER INQUIRIES INVITED.

COMPUTER DESIGN LABS
342 Columbus Avenue
Trenton, N.J. 08629

CIRCLE 95 ON READER SERVICE CARD

command is called, of all things, "loop" and the simplest form is:

```
loop
      statement1;
      statement2;
      .
      .
      statement-n;
end loop;
```

Obviously there is no way to get out of the above loop, so we will need a few more commands to make loops useful. The most important is the exit command which transfers control to the first statement after the end loop. For example:

```
I := 1;
loop
   if i = 10 then
        exit;
   end if;
   put(I);-- output statement to console device
   I := I + 1;
end loop;
```

The "if-then exit end if" sequence is a little bit cumbersome in this case, so Ada allows a more compact form of the exit command:

```
I := 1;
loop
   exit when I = 10;
   put(I);
   I := I + 1;
end loop;
```

Again we see how Ada syntax makes programs easier to read and more natural to write.

Like Pascal, Ada has a form of the while loop. It is:

```
while X < 10 loop
   -- statements;
end loop;
```

Once the value of X becomes less than 10, the loop terminates. We may also construct a loop similar to Pascal's repeat-until structure by proper placement of an exit statement:

```
loop
   -- statements;
      .
      .
   exit when I = 10;
end loop;
```

Since the test for leaving the loop takes place at the end of the loop, this control block acts just like the Pascal repeat-until.

We may naturally have loops within loops and it is here that the exit command really comes in handy. If a loop is given a label, then the exit statement may specify which loop it is to leave. For example:

```
XXX: while I /= 25 loop      -- "/=" means
   statement1;                  "not equal to"
   statement2;
   YYY: loop
      statement3;
      statement4;
      if J = 7 then
         statement5;
         exit XXX;           -- since there is a
                             -- label on the
      end if;                -- exit statement,
                             -- control is passed
end loop YYY;                -- to just after "end
statement6;                  -- loop XXX;"
   end loop XXX;
```

If a loop is given a label, then the corresponding end loop must also have that label.

Iteration

Ada also has iteration structures. Like Pascal's for I := 1 to 10 do, you may write in Ada:

```
for I in 1..10 loop
        -- statements;
end loop;
```

And you may go backwards as in most other languages:

```
for I in reverse 1..10 loop
        -- statements;
end loop;
```

but as in Pascal, you may only increment or decrement by 1.

This iteration scheme has one interesting feature not found in Pascal. The control variable need not (and may not) be declared in the program. So if we have:

```
I : integer;
I := 5;          -- I now has a value of five
```


```
for I in 7..21-- this is a new I of type
loop           -- "integer"
     .         -- since 7..21 is a range of
     .         -- integers

end loop;      -- I now has a value of 21
put(I);        -- this is the first "I". Five will
               -- be typed
               -- The "I" of the loop does not
               -- exist
               -- after the "end loop;"
```

This is a nice feature when working a lot with arrays since it is usually an inconvenience to have to declare indices every time you have to scan an array. But it also has its drawbacks. Suppose you wish to find the first element of a 6 by 6 array that is less than zero and get the indices of that element. The following will **not** work:

```
X : array (1..6,1..6) of integer;
I, J : integer;

OUTER: for I in 1..6 loop
  INNER: for J in 1..6 loop
    exit OUTER when X(I,J) < 0;
  end loop INNER;
end loop OUTER;
put(I); put(J);
            -- the correct indices will not be typed
```

Since the I and J in the loop are different from those declared outside the loop, the variables containing the indices cease to exist when control passes outside loop OUTER. The problem would have to be solved in the following manner:

```
X : array (1..6,1..6) of integer;
I, J : integer;

OUTER: for I1 in 1..6 loop
  INNER: for J1 in 1..6 loop
    if X(I1,J1) < 0 then
      I := I1;      -- put I1 and J1 in variables
                    -- that
      J := J1;      -- exist outside the loop.
     exit OUTER; -- now leave both loops
    end if;
  end loop INNER;
end loop OUTER;
```

If you are familiar with Pascal, you know that the control variable of an iteration need not be an integer. Likewise in Ada. If we define "CITY" as the following type:

```
type CITY is (CHICAGO, NEW_YORK,
BOSTON, LOS_ANGELES);
```

then the structure

```
for METRO in CHICAGO..LOS_ANGELES loop
        -- statements;
end loop;
```

is perfectly legal and METRO will be an undeclared variable of type CITY. As in most other languages, the control variable must be a discrete type (i.e., it can not be of type real).

The CASE Statement

Except for syntax, the Ada case statement is just like the case statement in Pascal. The following example will suffice to show how it is used:

```
type MONTH is (JAN,FEB,MAR, APR,
      MAY,JUN,JUL,AUG,SEP,OCT,NOV,
      DEC);
type SEASON is (SUMMER,FALL,WIN-
      TER,SPRING);
THIS_MONTH : MONTH;
THIS_SEASON : SEASON
HOLIDAY_TIME : boolean;
```

Note that in the Pascal standard, the case statement may not contain an else clause, even though most implementations of Pascal do include that feature.

The GOTO Statement

This much maligned statement has none of the restrictions found in Pascal. Actually, one of the most important uses of the goto is to transfer control out of deeply nested loops, and with Ada's exit command there is even less need for a goto than there is in Pascal. The destination of a goto is signified by a label in double angle brackets (<<LABEL>>) so that the label stands out. The use would be:

```
    goto DO_OVER;
        .
        .
        .
<<DO_OVER>>
    -- statements;
```

```
case THIS_MONTH is
   when JAN | FEB | MAR =  THIS_SEASON := WINTER;  -- "|" means "or"
   when APR | MAY | JUN =  THIS_SEASON := SPRING;
   when OCT | NOV | DEC =
                           THIS SEASON ;= FALL;
                           if THIS_MONTH = DEC then
                              HOLIDAY_TIME := true;
                           end if;
   when JUL | AUG | SEP => THIS_SEASON := SUMMER;


end case;
```

The case statement may also have a "when others" clause. Using the types and variables defined in the previous example and also

```
NUMBER_OF_DAYS : integer;
```

we have

```
case THIS_MONTH is
     when SEP | APR | JUN | NOV  => NUMBER_OF_DAYS := 30;
     when FEB                    => NUMBER_OF_DAYS := 28;
                                          -- forget leap year
     when others                 => NUMBER_OF_DAYS := 31;
end case;
```

You may not use a goto to transfer control to a label of a loop statement. Also, you may not transfer control into an if, case, or loop statement or into or out of a procedure or function.

BLOCKS

One feature of Ada that is entirely missing from Pascal is the block that is reached through the normal flow of a program. This feature does exist in PL/1. One purpose of the block is to bring variables into existence for a short period of time, as in:

```
        .          -- main program
        .
declare                -- declaration of local
                       -- variables
    TEMP : integer;
begin
                       -- block begins
      TEMP := A;
      A := B;
      B := TEMP;
end;                   -- TEMP disappears,
        .                 block ends
        .              -- main program continues
        .
```

Procedures

These most powerful structures of Pascal are naturally found in Ada. As an example of the syntax, let's look at a procedure which will multiply the elements of an array by a scalar (a single number). We will assume that the following type is defined globally in the package:

```
  type LIST is array(1..10) of integer;
```

The procedure is:

```
  procedure MULT_ARRAY (I : integer; X : in
            out LIST) is
            -- declare any constants, types, or
            -- variables local to
            -- the procedure here. There are
            -- none
            -- in this case.
    begin
        for J in 1..10 loop
            X(J) := X(J) * I;
        end loop;
  end MULT_ARRAY;
```

In the program the procedure would be invoked by a statement such as:

```
MULT_ARRAY(17, M); -- M is of type LIST
```

The unfamiliar item in the procedure is the mode of the parameter (in this case declaring LIST as "in out"). There are three parameter modes in Ada. They are "in", "out", and "in out". If the mode of the formal parameter is "in", then its value is provided by the actual parameter in the calling statement. The value may not be changed by the procedure, so in effect it becomes a parameter constant. The default mode is "in", so in the above example "I" is an "in" parameter. If a parameter is "out", then its value is assigned by the procedure but it cannot be read by the procedure. In the calling statement the actual parameter would have to be a variable since it is going to be assigned a value. An "in out" parameter may be both read by the procedure and assigned a value by the procedure, so the actual parameter would also have to be a variable. The "in out" mode is equivalent to the "var" mode of a parameter in Pascal.

For those not familiar with the terms "actual" and "formal" parameters, the actual parameters are those which are part of the calling statement (17 and M in the above example), while the formal parameters are those in the procedure (I and X in the above).

Ada also has a feature called "default" parameters. Let's say that most of the time we want to multiply our array by ten. We would then define our procedure as:

```
  procedure MULT_ARRAY (I : integer := 10;
            X in out LOOP) is
            -- everything else is the same as
            -- above
  end MULT_ARRAY;
```

The procedure may now be called by

```
  MULT_ARRAY(X => M);
            -- M is an array of type LIST
```

Since the first parameter is omitted, the value of "I" will take on its assigned default value of ten. But now the second parameter is out of position in the acutal parameter list, so a notation called "named notation" must be used. This associates the actual parameter M with the second formal parameter X. Of course we can still call MULT_ARRAY by:

```
  MULT_ARRAY(4, M);
```

in which case the default value of "I" is ignored. *The return from a procedure normally occurs when control reaches the end of the procedure block. However, if the return is desired when a*

Ada has a useful feature called "default" parameters.

certain condition occurs, the programmer may use the return *statement to return control to the calling statement.*

Overloading

Suppose we want to multiply each element of a twenty-element array by a constant. Obviously we can't use MULT_ARRAY since it expects the second parameter to be of type LIST which is array(1..10) of integer. Therefore we have to create another procedure. It will be:

```
type LIST2 is array(1..20) of integer;
procedure MULT_ARRAY(I : integer; X : in
          out LIST2) is
begin
  for J in 1..20 loop
        X(J) := X(J) * I;
  end loop;
end MULT_ARRAY;
```

But now we have two procedures called MULT_ARRAY. It's okay because Ada allows what is called "overloading." If we have two different variables

```
X10 : LIST;
X20 : LIST2;
```

then the procedure calls

```
MULT_ARRAY(7, X10);
```

and

```
MULT_ARRAY(7, X20);
```

will each call the correct procedure by first examining the types of the actual parameters and checking to see in which of the two procedures named "MULT_ARRAY" the actual parameters match the formal parameters. Of course if there is no procedure called "MULT_ARRAY" which matches the actual parameters, then we have an error.

Functions

Just like Pascal, Ada has subroutines called "functions." And like a Pascal function, an Ada function evaluates to a single value. Suppose we wish to add the corresponding elements of two structures of type LIST (array (1..10) of integer):

```
function ARRAY_SUM (A, B : LIST) return
LIST is
  C : LIST;          -- local variable for storing sum
begin
   for I in 1..10 loop
        C(I) := A(I) + B(I);
   end loop;
   return C;             -- the entire array may
                         -- be referred
end ARRAY_SUM;           -- to as "C"
```

If R, S, and T are variables of type LIST, then

```
R := ARRAY_SUM(S,T);
```

will give us the sum of S and T in R.

This leads us to a really great feature of Ada. Instead of calling the function "ARRAY_SUM", let's name the function "+":

```
function "+" (A, B : LIST) return LIST is -- quotes are necessary
   C : LIST;                              -- around the plus sign
begin
   for I in 1..10 loop
        C(I) := A(I) + B(I);              -- this plus sign is
   end loop;                              -- integer addition
end "+";
```

We can now get the sum of S and T in R by using

```
R := S + T;      -- no quotes around the plus
                 -- sign here
```

Thus we have overloaded the symbol "+". Ada will determine which addition must be done (our defined "+", predefined integer or real addition, or any other "+" that may be defined) by examining the operands (i.e., the actual parameters) and trying to match them with formal parameters of the same type.

This concludes Part I of the tutorial. In Part II, I shall discuss the various "types" that the programmer has or may create.

UCSD Adaptable Pascal Versions II.0 and IV.0

by John W. Moore, Robert G. Williams, and John R. Vidolich

We are in the chemistry department of Eastern Michigan University and are developing a microcomputer system for text processing, computer graphics, instrument interfacing, and computerized test generation. Our equipment includes a Cromemco ZPU and TU-ART digital interface, 64K of static RAM, a Houston Instruments digital plotter, a Cambridge Development Labs graphics interface, a Tecmar A/D Converter, and dot-matrix and daisy-wheel printers. Three Morrow Discus 2D floppy disk drives provide mass data storage. Not wishing to be limited to assembly language programming, we chose Pascal for its extraordinary clarity and structure. The UCSD implementation was selected since it could be run on our CP/M[1] system and is virtually processor-independent. UCSD Pascal also incorporates several powerful extensions to the Pascal defined by Jensen and Wirth, most notably string intrinsics, block input/output, and random access files. In addition, a complete operating system, including a powerful screen-oriented editor, a line-oriented editor, a file handler, Z80 and 8080 assemblers, a linker, and several other utilities are provided. We have found these to be very helpful in developing programs that collect and analyze data from a stopped-flow kinetics apparatus and in producing a library of routines for hard-copy and CRT graphics. This article describes our experiences in bringing up UCSD Pascal using the hardward described above.

Booting Standard UCSD Pascal, Version II.0, with our CP/M-based system was reasonably straightforward, though it did require three separate diskettes. Version II.0 is distributed on single-density diskettes, but since we were using Thinker Toys' double-density CP/M, it was necessary to boot CP/M with a double-density diskette, reset drive A to single density, and insert the single-density diskette containing the UCSD Pascal bootstrap loader (a CP/M .COM file). We could then complete the bootstrap by executing the loader program and inserting the Pascal system disk. We also encountered a minor problem in that the cursor-control features of our console terminal, a Microterm ACT-V, required control characters (such as CTL-P) that UCSD Pascal uses for other purposes and would not transmit to the terminal. Following Helmers' example[2] we solved this problem by setting the most significant bit of each cursor-control character, fooling Pascal but not the terminal, which looks only at the seven low-order bits. More recently another solution to this problem has been published,[3] but we have not tested it.

While we were quite pleased with Standard UCSD Pascal II.0, we did notice a few undesirable features. First, Pascal was available only on single-density diskettes and one of our applications, test generation, requires a large amount of disk storage. Clearly, a double-density Pascal is a desirable option. Furthermore, the Pascal II.0 operating system could access only two disk drives. Upon calling SofTech Microsystems, the licensed supplier for UCSD Pascal, we were told that an Adaptable Pascal, permitting double-density disk formats and access to more than two disk drives, had been developed. In February 1980 we received Adaptable Pascal, Version II.0, as a replacement for our standard version in exchange for reporting our experiences in using the new system.

The Adaptable System II.0 is distributed on four IBM-3740-compatible 8″ floppy diskettes, and consists of a CP/M disk, two Pascal disks, and a utilities disk. The Pascal disks are organized in three images (or parts), each small enough to fit on a 5¼″ minifloppy, and a DISKCHANGE program is provided to extract a given floppy-disk image. However, users having only 5¼″ disk drives will have to find a system capable of reading 8″ diskettes and writing 5¼″ diskettes, or they will not be able to convert the distribution diskettes to their format. The operating systems provided are the full Adaptable system, permitting multiple floppy disk formats and user-defined peripherals, and a limited version that uses CP/M's basic input/output system (BIOS). We decided to boot the CP/M Adaptable system first, since it promised to be a much easier task. Even so, we faced an immediate problem—the names on our diskettes didn't match the ones given in the manual! By a process of elimination we identified the Pascal sys-

Department of Chemistry, Eastern Michigan University, Ypsilanti, MI 48197

tem disk, booted CP/M, and executed the Pascal bootstrap (PASBOOT). This produced

UCSD Pascal (II.0) Booter version [A2]

Insert Pascal disk into drive A, then type<return> followed by

Reading secondary bootstrap

Booting to UCSD Pascal

Welcome CE20FDA

Date is . . .

at which point the system stopped. Subsequent attempts did no better, and a phone call to the helpful people at SofTech Microsystems led us to believe that the diskettes were preliminary versions that might have been written improperly, and so the diskettes were returned.

Ten days later, a package of four clearly labelled diskettes arrived. Again we followed the bootstrapping procedure, but got no further than "Booting to UCSD Pascal." However, before calling SofTech Microsystems we booted the system with CP/M, examined the PASBOOT source code, and found a conditional assembly directive enabling the booter to use CP/M's debugger, DDT. This option had originally been turned off (not selected), and so following instructions, we modified the bootstrap to run under DDT. Once again we tried bringing up the system, and this time were greeted with Pascal's welcoming message. CP/M Adaptable Pascal was up and running, but why was it necesssary to use DDT? Acting on a hunch, we used DDT to run the original PASBOOT program, and again successfully booted to Pascal. Apparently, the supplied PASBOOT. COM file had been assembled to run under DDT, contrary to what was indicated in the PASBOOT source file. SofTech Microsystems should warn users to reassemble PASBOOT, or at least specify whether it is to be run with or without DDT.

Next we tried booting Pascal without the aid of DDT by editing the source file and turning off the DDT option. Unfortunately, the system would not boot. Returning to the source file, we noticed that PASBOOT initialized its stack 14 bytes below the start of the CP/M BIOS. (The comment accompanying this code indicated that its purpose was to accommodate non-standard BIOSs.) Normally the BIOS is assumed to be entirely self-contained, but after much experimentation, we discovered that initializing the stack to 70 bytes below the BIOS allowed the system to boot. We concluded that the original location of the stack altered memory that was referenced by the BIOS, and that the BIOS provided by Thinker Toys was not exactly as described in the documentation supplied. The latter was essentially a copy of Digital Research's CP/M manuals.

We now had Pascal up and running, but bringing the system up required booting CP/M, executing the Pascal bootstrap, and swapping diskettes. Our next task was to produce a diskette that would boot directly to CP/M Adaptable Pascal. Track zero of this diskette was reserved for the bootstrap, consisting of:

1) CP/M's basic I/O system (BIOS) containing all I/O drivers;

2) A secondary bootstrap (SECBOOT) to read the Pascal interpreter into memory;

3) A primary bootstrap that would read the BIOS and secondary bootstrap into memory, load up the stack with parameters describing our system, and then jump to the secondary bootstrap to finish booting.

The adaptable Pascal diskettes included a sample primary bootstrap, SAMBOOT, which the user must complete by inserting assembly-language routines that transfer the BIOS and SECBOOT to the appropriate locations in memory. We were able to produce a working version of SAMBOOT (which we named PRIMARY) rather easily because of a very useful feature of the Morrow/Thinker Toys DISCUS 2D disk controller. All necessary software for selecting a disk drive, track, and sector and reading or writing from disk to memory are provided in a ROM on the controller card. These were accessed by calls from our bootstrap program, a much simpler programming task than writing our own disk-accessing routines. Once PRIMARY was ready (see Listing 1) we extracted a disk image containing SECBOOT from the system disk and copied it to a fresh diskette; all that remained was to transfer PRIMARY and the BIOS to track zero of the new diskette. A bootstrap copier, CPMBOOT, had been provided for this purpose, but when we used it, the resulting bootstrap diskette would not bring up the system. Again the problem was apparently that Thinker Toys' non-standard BIOS was not confined to the portion of memory where CPMBOOT expected to find it. A listing of the dual-density BIOS used by Thinker Toys would

UCSD incorporates several powerful extensions to the Pascal of Jensen and Wirth, most notably string intrinsics, block I/O and random access files.

have been helpful, but none was available. Fortunately Thinker Toys documentation provided a listing of a standard single-density BIOS, which we typed in and transferred to the bootstrapping disk. This time CP/M Adaptable Pascal booted successfully, leaving us with a one-diskette bootstrap but a far-from-optimal system. We still were limited to single density and could only access two of our three disk drives. So, using the knowledge gained from the CP/M Adaptable Pascal system, we turned to the full Adaptable Pascal system.

The full Adaptable UCSD Pascal system requires that the user write an SBIOS in assembly language for the appropriate microprocessor. The SBIOS consists of a jump table and a series of routines that test the status of, read from, or write to the console and disk drives. Using the routines in the Disk Jockey ROM and the newly written BIOS as a model, we wrote a Pascal SBIOS with little trouble. An SBIOS tester is provided to randomly read and write to all peripherals, printing diagnostics if any errors are detected. Surprisingly enough, the SBIOS worked properly the first time. The next task was to copy the SBIOS and PRIMARY to track zero of the bootstrapping disk. CPMBOOT provided with the CP/M Adaptable system worked nicely for this purpose. No troubles were encountered, and the system booted on the initial try.

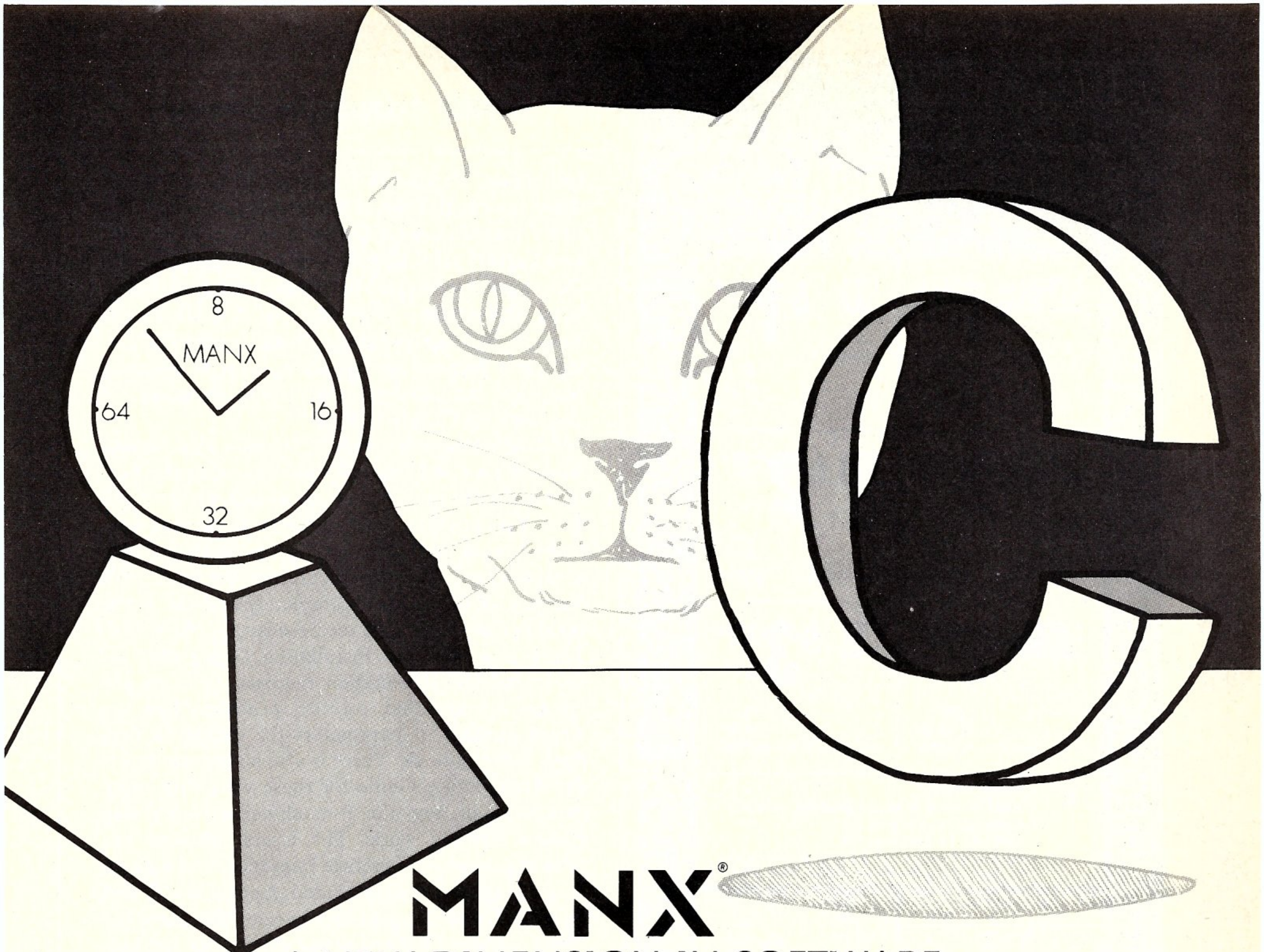
Buoyed by this success, we then implemented extensions to the SBIOS to permit multiple floppy disk formats and peripherals, such as a printer and digital plotter. UCSD doesn't supply a test program for the extended SBIOS, but this is not a serious problem. From our experience, anyone successfully booting the SBIOS should be able to write the extended SBIOS. After making sure our SBIOS met all the specifications given in the manual, we again used CPMBOOT to transfer it to the bootstrap disk. Once again, the system booted properly. Our Extended SBIOS is given in Listing 2.

We were now ready to create a double-density Adaptable Pascal diskette. At this point, we decided to write a Pascal program, FORMAT (see Listing 3), to alter the parameters of our disks, so that we might easily read a variety of disk formats. Using the newly written FORMAT program, we changed drive B to double density and transferred all necessary files with the Filer. However, we were unable to use CPMBOOT to alter the bootstrap. This was apparently related to the fact that we were now working with a disk drive that had been set to double density. We were able to solve the problem by returning to CP/M, loading the SBIOS and PRIMARY into core with DDT, then executing a program to copy from core to track 0 (see Listing 4). We then booted double-density UCSD Pascal.

Our experience in implementing the UCSD Pascal Adaptable System, Version II.0, was, in general, quite rewarding, though we do have a few complaints. Portions of the instruction manual were troublesome; at one point we were referred to a non-existent appendix, and it was sometimes unclear whether instructions refer to the CP/M Adaptable disk or the Adaptable system disk. One often has to read as many as three sections of the manual at once to understand a particular point. An example of this is the list of parameters to be loaded on the stack by the primary bootstrap. These actually are given in the section on the SBIOS tester, and, at least for 8080/Z80 systems, one parameter too many is listed in the documentation. Also, losing one's place in the manual can be disastrous, because the index is less than complete. Several readings and lots of page turning are required before everything becomes clear. Had we fully understood from the start what we eventually gleaned from the UCSD Pascal manual, we probably could have produced our SBIOS for the full adaptable system in less than a week. However, it took much longer to boot the different versions described above and to figure out exactly what had to be done. The inexperienced programmer should beware, since implementing the bootstrap requires that the reader be fluent in assembly language, but bootstrapping Adaptable Pascal is an excellent way to learn how your computer system works.

Recently SofTech Microsystems has introduced the UCSD P-System[1] and UCSD Pascal Version IV.0. This version offers upward-compatibility of source code from other versions, although recompilation of Version II.0 programs is required because the pseudo-code contains some new instructions. Version IV.0 also introduces multitasking and provides more flexible memory-management

The User's Manual has been expanded, its typeface improved, and a much better index has been provided.



MANX®

A NEW DIMENSION IN SOFTWARE

For Quality, Performance, Price, and Features *There Is No Comparison*

C COMPILERS - COMMON FEATURES:

• UNIX VER 7 compatibility (Full implementation of "C" as defined in "The C Programming Language" by Brian W. Kernighan and Dennis M. Ritchie.") • standard float, double, and long support • run time library with full I/O and source • fast compilation and execution • ALL C Operators • supports assembler subroutines • supports modular programming with true extern support • extern, static, auto, and register storage classes • unions, structures, multi-dimension arrays • short and unsigned data types • if, while, do, for, switch, case, goto • #define, #include • #ifdef, #ifndef, #else, #endif • command line arguments using argc and argv • DATA FORMATS: char- 8 bits • short- 16 bits • long- 32 bits • float- 32 bits • double- 64 bits.

AZTEC C II CP/M (MP/M) **$199**

NEW!!! Overlays • Library utility • produces relocatable 8080 source code (optional Z80 instructions) • relocating assembler and linker supplied • MICROSOFT M80 interface • SID/ZSID interface • requires 48K TPA (Apple requires 16K card). **(Also available for HEATH HDOS).**

AZTEC C II APPLE DOS **$199**

• relocating assembler and linker supplies • APPLE SHELL • VED editor • library and other utilities • requires 16K card.

C86 IBM PC • MSDOS • CP/M-86 **$249**

• directly produces 8088/8086 code • linker supplied

K & R BOOK . . $18 Manuals $30

VISA

ORDER BY PHONE OR MAIL

Specify products and disk format

master charge

CP/M FORMATS: 8"STD, HEATH, APPLE, OSBORNE, NORTHSTAR,...

OUTSIDE U.S.A. - Add $10 In N.J. add 5% sales tax

MANX®

software systems

Box 55, Shrewsbury, N.J. 07701
(201) 780-4004

CIRCLE 64 ON READER SERVICE CARD

techniques. Version IV.0 incorporates and consolidates the capabilities of Version II.0, Version II.1 (the Apple Computer Corporation UCSD Pascal), and Version III.0 (distributed by Western Digital Corporation with the Microengine hardware-emulated P-machine). We obtained UCSD Pascal Version IV.0 as an upgrade to our Version II.0 system, and because we have a number of Fortram programs available on a mainframe computer, we also ordered UCSD Fortran.

UCSD Adaptable Pascal, Version IV.0, is distributed on five 8″ floppy diskettes, with the Fortran compiler and library on a sixth diskette. As with Version II.0, each diskette is divided into three images, each small enough to be loaded onto a minifloppy. The documentation provided with Version IV.0 has been completely redone and is far superior to what we received with Version II.0. There is a separate Installation Guide that includes all the information required to bring the system up and that corrects problems like the wrong number of parameters for PRIMARY to push onto the stack. There is also a separate Internal Architecture Guide for those who want more details about how the P-machine and operating system actually work. The Users' Manual has been expanded, its typeface improved, and a much better index has been provided. Appendices G and H in the Users' Manual summarize differences between Version IV.0 and previous versions and provide guidelines for converting Pascal programs to Version IV.0. Our only quibble about the documentation is that for users like us, who have implemented previous versions of the Adaptable System, it would have been useful to have a one- or two-page summary of differences in the bootstrapping procedure. Only minor changes are required, but they are not spelled out explicitly.

Booting Version IV.0 was straightforward, given our previous experience. We used the FORMAT program (Listing 3) on our Version II.0 system to set disk parameters so that we could read the distribution diskettes and then used the Version II.0 filer to transfer the necessary files to a clean diskette. Then we copied our Version II.0 bootstrap to track zero and tried the new diskette. No go! We tried again using the secondary bootstrap supplied with Version IV.0, but still no welcoming message. Finally we sat down with the old and new manuals side by side and compared them, a process that revealed that the new primary bootstrap had to push one more parameter onto the stack. This has been accommodated by a conditional assembly directive in Listing 1, and this new

II.0 and IV.0 continued . . .

bootstrap brought the system up.

The manual for UCSD Pascal Version IV.0 states that source programs from Versions II.0, II.1, and III.0 should compile and *most* will run. However, the Version IV.0 system occupies more memory than previous versions and so "tight-fitting" programs might not run. This can be overcome by using segment procedures, but we preferred not to modify our existing programs. Consequently we wanted to make optimal use of the 64K of memory in our system. The UCSD P-system requires memory space for the p-code interpreter, for the SBIOS, and for a user program. In our system the DJ 2D ROM on the Morrow disk controller is addressed at F000 hex, and so a large contiguous memory area is available from 0000 hex to F000 hex. In Version II.0 we put both the interpreter and the SBIOS in this space. The DJ 2D board contains ROM from F000 to F3FF hex, and RAM from F400 to F7FF hex. Since our Seattle Products RAM can only be addressed in 4K blocks and since the RAM must not overlap the DJ 2D ROM, we cannot use memory above F7FF hex. The primary bootstrap is loaded at F700 hex, but 300 hex of RAM is available from F400 to F6FF hex. Since our SBIOS occupied less than 200 hex bytes, we decided to load the SBIOS into the DJ 2D RAM at F400 hex. This leaves 100 hex bytes for possible future expansion of the SBIOS and adds 200 hex bytes to the large contiguous RAM area for user programs. So far we have not encountered problems with tight-fitting programs, although we have run out of memory when using the screen editor. It would be very useful if the screen editor or YALOE were able to page text from the disk instead of requiring that the entire file to be edited must be loaded into memory.

We were very happy with UCSD Pascal Version II.0 and we used it to develop an extensive graphics package for our Cambridge Development Laboratory screen graphics, for screen dumps to an IDS 440 Pager Tiger dot-matrix printer, and for plotting on a Houston instruments DMP-2 plotter. Development of programs was very convenient because of the screen-oriented editor and the inherent structure of Pascal. The ability to write assembly-language routines, assemble them, and store them in the system library greatly aided program development, too. Once we had the UCSD Pascal system up and running, the graphics package as well as an applications package for collecting data from scientific instruments were developed very easily and rapidly (within three months).

UCSD Pascal IV.0 is an improvement on an already excellent software package, and we are happy with our decision to upgrade. However, the improvements in the new version are not as obvious yet as are some of the problems we have encountered. This is probably because we have only had a short time to get used to Version IV.0. We discovered that after recompiling all of our library routines for the graphics package we could no longer store them as we had done with Version II.0, but instead we had to link together all assembly-language and Pascal routines into a single unit before storing the unit in the system library. Version II.0 was more flexible in this regard, and we were unable to figure out from the Users' Manual why Version IV.0 seemed unable to find our graphics package. Several calls to SofTech Microsystems were required before we were able to contact the right person to answer this question. Another disappointment was our discovery in the *Fortran User Reference Manual* that "It is not generally possible to do I/O from Pascal routines called from a main program that is written in Fortran." This is a major problem for us because our graphics package is in Pascal and assembler, and many of the Fortran programs we would like to use include graphics. Thus we are faced with the problem of rewriting the graphics package in Fortran, eliminating I/O from the graphics package, or rewriting all our Fortran programs in Pascal.

All of this goes to show that the UCSD P-System is not perfect. Nevertheless, it is an excellent operating system with many powerful features, and we recommend it highly. We hope that the information and listings presented here will be useful to others who plan to implement the UCSD P-System on 8080- or Z80-based machines.

References

1. CP/M is a registered trademark of Digital Research; UCSD Pascal and UCSD P-System are registered trademarks of the Regents of the University of California.
2. C. Helmers, *Byte*, 5 (2), 2–10 (February 1980).
3. B. Franks, *Byte*, 5 (7), 17 (July 1980).

Listing 1

```
                  **************************************************************
                  *                                                            *
                  *        PRIMARY BOOTSTRAP FOR UCSD ADAPTABLE PASCAL         *
                  *                                                            *
                  *   Whenever a cold boot (reset) occurs, the Disk Jockey     *
                  *  disk controller reads the contents of track 0, sector 1   *
                  *  into memory starting at ORIGIN+700H and executes it.      *
                  *  PRIMARY is designed to run at ORIGIN+700H, and automati-  *
                  *  cally boots to UCSD Pascal by reading the I/O drivers     *
                  *  (SBIOS) and secondary bootstrap (SECBOOT) from track 0    *
                  *  of the bootstrapping disk and loading them into memory.   *
                  *  It then jumps to SECBOOT to complete the bootstrapping    *
                  *  procedure. PRIMARY must reside on the first sector of     *
                  *  track 0 on the bootstrapping disk.                        *
                  *                                                            *
                  **************************************************************

FFFF =            TRUE          EQU    0FFFFH          ;DEFINE VALUE OF TRUE
0000 =            FALSE         EQU    NOT TRUE        ;DEFINE VALUE OF FALSE
FFFF =            PAS$IV        EQU    TRUE            ;SET THIS TRUE IF USING UCSD
                                                       ;  P-SYSTEM VERSION IV.0.
                                                       ;  SET FALSE FOR VER. II.0.

F000 =            ORIGIN        EQU    0F000H          ; ADDRESS OF DJ CONTROLLER BOARD
F400 =            SBIOS         EQU    0F400H          ; ADDRESS OF THE SBIOS
F009 =            TKZERO        EQU    ORIGIN+11Q      ;TRACK ZERO SEEK ENTRY POINT
F00C =            TRKSET        EQU    ORIGIN+14Q      ;ENTRY FOR TRACK SEEK
F00F =            SETSEC        EQU    ORIGIN+17Q      ;ENTRY POINT FOR SECTOR SET
F012 =            SETDMA        EQU    ORIGIN+22Q      ;ENTRY POINT TO SET READ/WRITE ADDRESS
F015 =            DREAD         EQU    ORIGIN+25Q      ;DISK READ ENTRY POINT
F018 =            DWRITE        EQU    ORIGIN+30Q      ;DISK WRITE ROUTINE ADDRESS
F024 =            DMAST         EQU    ORIGIN+44Q      ;DISK READ/WRITE STATUS ROUTINE
0010 =            SECNUM        EQU    16              ; SECONDARY BOOTSTRAP IS 16 SECTORS LONG
0003 =            SECSEC        EQU    3               ; SECONDARY BOOTSTRAP ON THIS SECTOR
0005 =            BIOSNUM       EQU    5               ; SBIOS IS 5 SECTORS LONG
0013 =            BIOSSEC       EQU    19              ; SBIOS IS ON THIS SECTOR
8200 =            SECBOOT       EQU    8200H           ;SECONDARY BOOTSTRAP RUNS HERE

0100 =            INTERP$BASE   EQU    100H            ;FIRST LOC USED BY INTERPRETER
0100 =            MAX$BYTES     EQU    256             ;MAXIMUM NO. BYTES PER SECTOR
EFFE =            TOP$MEMORY    EQU    ORIGIN-2        ;TOP WORD OF CONTIGUOUS MEMORY
0100 =            LOW$MEMORY    EQU    100H            ;LOWEST AVAILABLE RAM LOCATION
004D =            TRACKS        EQU    77              ;NUMBER OF TRACKS PER DISK
001A =            SECTORS       EQU    26              ;NUMBER OF SECTORS PER TRACK
0100 =            BYTES         EQU    256             ;NUMBER OF BYTES PER SECTOR
0006 =            INTERLEAVE    EQU    6               ;INTERLEAVING FACTOR
0001 =            FIRST$TRACK   EQU    1               ;FIRST INTERLEAVED TRACK
0000 =            SKEW          EQU    0               ;TRACK-TO-TRACK SKEW
001A =            MAX$SECTORS   EQU    26              ;MAXIMUM NUMBER OF SECTORS/TRACK

F000 =            STACK         EQU    TOP$MEMORY+2    ;INITIAL STACK POINTER

                  **********************************
                  *  PRIMARY BOOTSTRAP STARTS HERE  *
                  **********************************

F700                      ORG    ORIGIN+700H
F700 2100F0       PBOOT   LXI    H,STACK            ; STACK AT TOP OF CONTIG. MEMORY
F703 F9                   SPHL                      ; RESET THE STACK
F704 2100F4               LXI    H,SBIOS            ;SBIOS GOES HERE
F707 011305               LXI    B,(BIOSNUM SHL 8)+BIOSSEC ; B - # OF SECTORS TO READ
                                                    ; C - STARTING SECTOR
F70A CD46F7               CALL   READIT             ; READ IN CBIOS
F70D 210082               LXI    H,SECBOOT          ; LOAD BOOT BASE ADDRESS
F710 010310               LXI    B,(SECNUM SHL 8)+SECSEC ;B - # OF SECTORS TO READ
                                                    ; C - STARTING SECTOR
F713 CD46F7               CALL   READIT             ; READ IN SECONDARY BOOTSTRAP
```

```
                      IF      PAS$IV          ;PUSH ON STACK ONLY IF PASCAL VER.
                                              ;  IV.0 IS BEING BOOTED
F716 210001           LXI     H,MAX$BYTES     ;MAX BYTES/SECTOR OF ANY DISK
F719 E5               PUSH    H
                      ENDIF

F71A 211A00           LXI     H,MAX$SECTORS   ;MAX SECTORS IN TABLE
F71D E5               PUSH    H
F71E 210000           LXI     H,SKEW          ;TRACK-TO-TRACK SKEW
F721 E5               PUSH    H
F722 210100           LXI     H,FIRST$TRACK   ;FIRST PASCAL TRACK
F725 E5               PUSH    H
F726 210600           LXI     H,INTERLEAVE    ;INTERLEAVING FACTOR
F729 E5               PUSH    H
F72A 210001           LXI     H,BYTES         ;BYTES PER SECTOR
F72D E5               PUSH    H
F72E 211A00           LXI     H,SECTORS       ;SECTORS PER TRACK
F731 E5               PUSH    H
F732 214D00           LXI     H,TRACKS        ;TRACKS PER DISK
F735 E5               PUSH    H
F736 21FEEF           LXI     H,TOP$MEMORY    ;TOP OF AVAILABLE MEMORY
F739 E5               PUSH    H
F73A 210001           LXI     H,LOW$MEMORY    ;BOTTOM OF AVAILABLE MEMORY
F73D E5               PUSH    H
F73E 1100F4           LXI     D,SBIOS         ;ADDRESS OF SBIOS
F741 D5               PUSH    D
F742 E5               PUSH    H               ;ADDRESS OF INTERPRETER
F743 C30082           JMP     SECBOOT


             ***************************************************************
             *                                                             *
             *  READIT: READ THE NUMBER OF SECTORS SPECIFIED IN THE B REG, *
             *          STARTING AT THE SECTOR SPECIFIED IN THE C REG, INTO *
             *          THE MEMORY LOCATION SPECIFIED IN THE HL PAIR.      *
             *                                                             *
             ***************************************************************

F746 C5      READIT   PUSH    B               ;SAVE SECTOR AND COUNT
F747 E5               PUSH    H               ;SAVE STARTING ADDRESS
F748 CD0FF0           CALL    SETSEC          ;SET SECTOR TO READ
F74B CD09F0           CALL    TKZERO          ;HOME THE DRIVE
F74E E1               POP     H               ;GET STARTING LOAD ADDRESS
F74F 44      LDLOOP:  MOV     B,H             ;PUT STARTING ADDRESS IN BC
F750 4D               MOV     C,L
F751 CD12F0           CALL    SETDMA          ;SET LOAD ADDRESS
F754 060A             MVI     B,10            ;INITIALIZE RETRY COUNTER
F756 C5      RDLOOP:  PUSH    B               ;SAVE RETRY COUNTER
F757 CD15F0           CALL    DREAD           ;READ THE SECTOR
F75A C1               POP     B               ;FETCH RETRY COUNTER
F75B D265F7           JNC     RDGOOD          ;JUMP IF READ OK
F75E 05               DCR     B               ;ELSE UPDATE RETRY COUNTER
F75F C256F7           JNZ     RDLOOP          ;TRY AGAIN IF <10 ERRORS
F762 C300F0  EXIT:    JMP     ORIGIN          ;START OVER - TOO MANY ERRORS
F765 C1      RDGOOD:  POP     B               ;GET SECTOR COUNT AND NUMBER
F766 05               DCR     B               ;UPDATE SECTOR COUNT
F767 C8               RZ                      ;RETURN IF DONE
F768 0C               INR     C               ;INCREMENT SECTOR NUMBER
F769 C5               PUSH    B               ;SAVE COUNT AND NUMBER
F76A CD0FF0           CALL    SETSEC          ;GET SECTOR TO READ
F76D CD24F0           CALL    DMAST           ;GET LOAD ADDRESS
F770 218000           LXI     H,128           ;UPDATE IT
F773 09               DAD     B
F774 C34FF7           JMP     LDLOOP          ;READ NEXT SECTOR
```

Lisitng 2

```
         ***************************************************************
         *                                                             *
         *            EXTENDED SBIOS FOR UCSD ADAPTABLE PASCAL         *
         *                                                             *
         *                                                             *
         *  WRITTEN:      MAY  1980        RGW                         *
         *                                                             *
         *  MODS:                                                      *
         *                                                             *
         *   7/15/80      RGW              USE PAPER TIGER AS PRINTER  *
         *   7/5/81       JWM              SET ORIGIN AT F400H         *
         *                                                             *
         ***************************************************************
         *                                                             *
         *  CONSOLE:     CRT ON SERIAL A (9600 BAUD)                   *
         *                                                             *
         *  PRINTER:     PAPER TIGER ON PARALLEL B                     *
         *                                                             *
         *  REMOTE UNIT: DIGITAL PLOTTER ON DISK JOCKEY SERIAL PORT    *
         *                                                             *
         ***************************************************************

         ******************************
         *  SYSTEM EQUATES            *
         ******************************

F000 =   ORIGIN  EQU  0F000H             ;DISK JOCKEY/2D CONTROLLER STARTING ADDRESS
F400 =   BIOS    EQU  0F400H             ;LOCATION OF EXTENDED SBIOS

         **************************************************************
         *  THINKER TOY I/O ROUTINES FOR THE UART ON THE DISK JOCKEY-2  *
         **************************************************************

F003 =   REMIN   EQU    ORIGIN+3         ;SERIAL INPUT ROUTINE
F006 =   REMOUT  EQU    ORIGIN+6         ;SERIAL OUTPUT ROUTINE
F009 =   TKZERO  EQU    ORIGIN+9H        ;TRACK ZERO SEEK ROUTINE
F00C =   TRKSET  EQU    ORIGIN+0CH       ;REGULAR TRACK SEEK ROUTINE
F00F =   SETSEC  EQU    ORIGIN+0FH       ;SET SECTOR ROUTINE
F012 =   SETDMA  EQU    ORIGIN+12H       ;READ/WRITE BEGINNING ADDRESS SET
F015 =   DREAD   EQU    ORIGIN+15H       ;DISK READ ROUTINE
F018 =   DWRITE  EQU    ORIGIN+18H       ;DISK WRITE ROUTINE
F01B =   SELDRV  EQU    ORIGIN+1BH       ;DISK SELECTION ROUTINE
F021 =   TSTAT   EQU    ORIGIN+21H       ;SERIAL DEVICE STATUS ROUTINE
F640 =   STACK   EQU    ORIGIN+640H      ;DISK JOCKEY/2D RAM AREA FOR BOOT ONLY
0099 =   SEKERR  EQU    99H              ;SEEK ERROR BIT MASK
00FF =   RWERR   EQU    0FFH             ;READ/WRITE ERROR BIT MASK
000D =   ACR     EQU    0DH              ;CARRIAGE RETURN
000A =   ALF     EQU    0AH              ;LINE FEED

         ***********************************
         *  CROMEMCO TUART PORT ADDRESSES  *
         ***********************************

0020 =   ABASE   EQU  20H                ;BASE ADDRESS FOR T-UART DEVICE A
0080 =   BBASE   EQU  80H                ;BASE ADDRESS FOR T-UART DEVICE B
0020 =   ASTATPT EQU  ABASE              ;STATUS AND BAUD RATE PORT A
0080 =   BSTATPT EQU  BBASE              ;STATUS AND BAUD RATE PORT B
0021 =   ASERPT  EQU  ABASE+1            ;INPUT & OUTPUT SERIAL PORT A
0081 =   BSERPT  EQU  BBASE+1            ;INPUT & OUTPUT SERIAL PORT B
0022 =   ACMDPT  EQU  ABASE+2            ;COMMAND PORT FOR A
0082 =   BCMDPT  EQU  BBASE+2            ;COMMAND PORT FOR B
0023 =   AINTPT  EQU  ABASE+3            ;INTERRUPT PORT A
0083 =   BINTPT  EQU  BBASE+3            ;INTERRUPT PORT B
0024 =   APARPT  EQU  ABASE+4            ;INPUT & OUTPUT PARALLEL PORT A
0084 =   BPARPT  EQU  BBASE+4            ;INPUT & OUTPUT PARALLEL PORT B
```

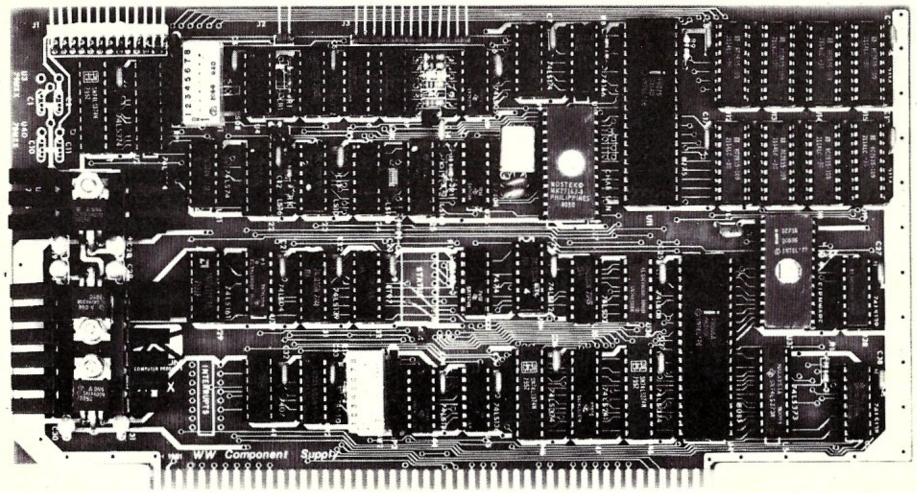
```
                 ***************
                 *  CONSTANTS  *
                 ***************

0040 =           RXRDY   EQU  40H               ;READ DATA AVAILABLE (READY TO READ)
0080 =           TXRDY   EQU  80H               ;TRANSMIT BUFFER EMPTY (READY TO TRANSMIT)

00C0 =           BR9600  EQU  0C0H              ;BAUD RATE OF 9600, 1 STOP BIT
0084 =           BR300   EQU  84H               ;BAUD RATE OF  300, 1 STOP BIT
0001 =           RESET   EQU  1                 ;RESET THE TUART,LOW BAUD RATE,NO INTERRUPTS

0007 =           BELL    EQU  7H                ;BELL
0008 =           BS      EQU  08H               ;BACKSPACE
0009 =           HT      EQU  09H               ;HORIZONTAL TAB
000A =           LF      EQU  0AH               ;LINE FEED
000B =           VT      EQU  0BH               ;VERTICAL TAB
000C =           FF      EQU  0CH               ;FORM-FEED
000D =           CR      EQU  0DH               ;CARRIAGE RETURN
001B =           ESC     EQU  1BH               ;ESCAPE
0020 =           SPC     EQU  20H               ;SPACE
007F =           RUBOUT  EQU  7FH               ;RUBOUT
0000 =           MEND    EQU  00H               ;END OF MESSAGE INDICATOR

007F =           ASCII   EQU  7FH               ;ASCII MASK


                 **********************
                 *  SBIOS STARTS HERE  *
                 **********************


F400                     ORG     BIOS

F400 C35CF4              JMP     SYSINIT        ;INITIALIZE SYSTEM
F403 C376F4              JMP     SYSHALT        ;EXIT UCSD PASCAL
F406 C377F4              JMP     CONINIT        ;CONSOLE INITIALIZATION
F409 C379F4              JMP     CONSTAT        ;CONSOLE STATUS
F40C C383F4              JMP     CONREAD        ;CONSOLE INPUT
F40F C38FF4              JMP     CONWRIT        ;CONSOLE OUTPUT
F412 C39BF4              JMP     SETDISK        ;SET DISK NUMBER
F415 C3EDF4              JMP     SETTRAK        ;SET TRACK NUMBER
F418 C3F2F4              JMP     SETSECT        ;SET SECTOR NUMBER
F41B C3F7F4              JMP     SETBUFR        ;SET BUFFER ADDRESS
F41E C3FEF4              JMP     DSKREAD        ;READ SECTOR FROM DISK
F421 C339F5              JMP     DSKWRIT        ;WRITE SECTOR TO DISK
F424 C371F5              JMP     DSKINIT        ;RESET DISK
F427 C373F5              JMP     DSKSTRT        ;ACTIVATE DISK
F42A C374F5              JMP     DSKSTOP        ;DEACTIVATE DISK
F42D C375F5              JMP     PRNINIT        ;PRINTER INITIALIZATION
F430 C377F5              JMP     PRNSTAT        ;PRINTER STATUS
F433 C37BF5              JMP     PRNREAD        ;PRINTER INPUT
F436 C37DF5              JMP     PRNWRIT        ;PRINTER OUTPUT
F439 C3C1F5              JMP     REMINIT        ;REMOTE INITIALIZATION
F43C C3C3F5              JMP     REMSTAT        ;REMOTE STATUS
F43F C3CDF5              JMP     REMREAD        ;REMOTE INPUT
F442 C3D3F5              JMP     REMWRIT        ;REMOTE OUTPUT
F445 C3D8F5              JMP     USRINIT        ;USER DEVICES INITIALIZATION
F448 C3DBF5              JMP     USRSTAT        ;USER DEVICES STATUS
F44B C3E2F5              JMP     USRREAD        ;USER DEVICES INPUT
F44E C3EBF5              JMP     USRWRIT        ;USER DEVICES OUTPUT
F451 C3F4F5              JMP     CLKREAD        ;SYSTEM CLOCK READ

F454 0000        CURDISK DB      0,0            ;CURRENT FLOPPY DISK DRIVE NUMBER
F456 0000        CURTRAK DB      0,0            ;CURRENT TRACK ON CURDISK
F458 0000        CURSECT DB      0,0            ;CURRENT SECTOR ON CURTRAK
F45A 0000        CURBUFR DB      0,0            ;CURRENT MEMORY BUFFER ADDRESS
```

```
             **********************************************************
             *                                                        *
             * SYSINIT: SYSTEM INITIALIZATION ROUTINE                 *
             *                                                        *
             *         GIVEN  BC = POINTER TO JUMP TABLE              *
             *                                                        *
             **********************************************************

F45C 21EBF4  SYSINIT LXI     H,JMPTBL       ;SAVE POINTER TO INTERPRETER JUMP TABLE
F45F 71              MOV     M,C            ;USED BY SETDISK TO CALL DSKCHNG
F460 23              INX     H
F461 70              MOV     M,B
F462 3E01            MVI     A,RESET        ;RESET THE T-UART
F464 D322            OUT     ACMDPT         ;RESET PORT A
F466 D382            OUT     BCMDPT         ;RESET PORT B
F468 AF              XRA     A              ;NO INTERRUPTS PLEASE!!!
F469 D383            OUT     BINTPT         ;FOR THE B PORT NO INTERRUPTS
F46B D323            OUT     AINTPT         ;THE A PORT TOO
F46D 3EC0            MVI     A,BR9600       ;BAUD RATE OF 9600, 1 STOP BIT
F46F D320            OUT     ASTATPT        ;SET THE CONSOLE TO 9600 BAUD
F471 3E84            MVI     A,BR300        ;BAUD RATE OF 300, 1 STOP BIT
F473 D380            OUT     BSTATPT        ;SET SERIAL B TO 300 BAUD
F475 C9              RET


             *********************************************************
             *                                                       *
             * SYSHALT: EXIT UCSD PASCAL.                            *
             *                                                       *
             *********************************************************

F476 76      SYSHALT HLT                    ;HALT


             *********************************************************
             *                                                       *
             * CONINIT: INITIALIZE CONSOLE PORT AND REPORT STATUS OF *
             *          THE CONNECTION.                              *
             *                                                       *
             *          RETURNS A = 0 IF KEYBOARD ON LINE            *
             *                    = 9 IF KEYBOARD OFF LINE           *
             *                                                       *
             *********************************************************

F477 AF      CONINIT XRA     A              ;CONSOLE ON LINE
F478 C9              RET


             *********************************************************
             *                                                       *
             * CONSTAT: REPORT THE STATUS OF THE CONSOLE CONNECTION  *
             *          AND CONSOLE INPUT CHANNEL.                   *
             *                                                       *
             *          RETURNS A = 0 IF KEYBOARD ON LINE            *
             *                    = 9 IF KEYBOARD OFF LINE           *
             *                  C = 0 IF NO CHARACTER                *
             *                    = FF IF CHARACTER AVAILABLE        *
             *                                                       *
             *********************************************************

F479 DB20    CONSTAT IN      ASTATPT        ;GET STATUS
F47B E640            ANI     RXRDY          ;MASK FOR DATA READY BIT
F47D 4F              MOV     C,A            ;ZERO IN C FOR POSSIBLE RETURN
F47E C8              RZ                     ;SIGNAL ON LINE BUT NO CHARACTER
F47F AF              XRA     A              ;ZERO IN A REGISTER
F480 0EFF            MVI     C,0FFH         ;THERE IS A CHARACTER
F482 C9              RET
```

```
            ************************************************************
            *                                                          *
            * CONREAD: RECEIVE CHARACTER FROM CONSOLE AND REPORT       *
            *          STATUS OF CONNECTION.                           *
            *                                                          *
            *          RETURNS A = 0 IF OK                             *
            *                    = 1 IF ERROR                          *
            *                    = 9 IF OFF LINE                       *
            *                  C = INPUT CHARACTER                     *
            *                                                          *
            ************************************************************

F483 DB20   CONREAD IN      ASTATPT         ;GET THE I/O STATUS OF PORT A
F485 E640           ANI     RXRDY           ;MASK THE READ DATA AVAILABLE BIT
F487 CA83F4         JZ      CONREAD         ;LOOP UNTIL READY
F48A DB21           IN      ASERPT          ;GET THE DATA
F48C 4F             MOV     C,A             ;STORE CHARACTER IN C
F48D AF             XRA     A               ;REPORT CONSOLE ON LINE
F48E C9             RET

            ************************************************************
            *                                                          *
            * CONWRIT: OUTPUT A CHARACTER TO THE CONSOLE AND REPORT    *
            *          STATUS OF THE CONNECTION.                       *
            *                                                          *
            *          GIVEN   C = OUTPUT CHARACTER                    *
            *          RETURNS A = 0 IF OK                             *
            *                    = 9 IF OFF LINE                       *
            *                                                          *
            ************************************************************

F48F DB20   CONWRIT IN      ASTATPT         ;GET THE STATUS
F491 E680           ANI     TXRDY           ;MASK THE TRANSMIT BUFFER EMPTY BIT
F493 CA8FF4         JZ      CONWRIT         ;LOOP UNTIL READY TO OUTPUT
F496 79             MOV     A,C             ;GET THE DATA TO OUTPUT
F497 D321           OUT     ASERPT          ;OUTPUT TO SERIAL A
F499 AF             XRA     A               ;REPORT CONSOLE ON LINE
F49A C9             RET

            ************************************************************
            *                                                          *
            * SETDISK: ROUTINE TO SET CURRENT DISK (CURDISK).          *
            *                                                          *
            *          GIVEN  C = DISK NUMBER                          *
            *                                                          *
            ************************************************************

F49B 3A54F4 SETDISK LDA     CURDISK         ;CHANGING DRIVES?
F49E B9             CMP     C
F49F C8             RZ                      ;RETURN IF NO CHANGE
F4A0 79             MOV     A,C
F4A1 FE04           CPI     4               ;LEGAL DRIVE NUMBER? (0-3 ARE PERMISSIBLE)
F4A3 F0             RP                      ;RETURN IF NOT.
F4A4 3254F4         STA     CURDISK         ;STORE NEW DRIVE NUMBER IN CURDISK
F4A7 21AFF4         LXI     H,DSKINFO-12    ;COMPUTE POINTER TO CURDISK PARAMETERS
F4AA 010C00         LXI     B,12
F4AD 09     DADLOOP DAD     B
F4AE 3D             DCR     A
F4AF F2ADF4         JP      DADLOOP
F4B2 E5             PUSH    H               ;STORE IN BC PAIR
F4B3 C1             POP     B
F4B4 2AEBF4         LHLD    JMPTBL          ;HL CONTAINS POINTER TO INTERPRETER JUMP TABLE
F4B7 23             INX     H               ;WANT TO JUMP TO 2ND ENTRY IN THE JUMP TABLE
F4B8 23             INX     H               ;SO ADD 3 (THE LENGTH OF 1 JUMP INSTRUCTION)
F4B9 23             INX     H
F4BA E9             PCHL                    ;JUMP TO DSKCHNG. IT WILL RETURN TO CALLER

            ************************************************************
            *                                                          *
            * DSKINFO: FOUR DISK INFORMATION BLOCKS CONTAINING THE     *
            *          RECORDING FORMATS OF ALL FLOPPY DISK DRIVES.    *
            *          EACH BLOCK CONTAINS SIX 16-BIT WORDS:           *
            *                                                          *
            *          WORD            DEFINITION                      *
            *           0          NUMBER OF TRACKS PER DISK           *
            *           1          NUMBER OF SECTORS PER TRACK         *
            *           2          NUMBER OF BYTES PER SECTOR          *
            *           3          INTERLEAVING FACTOR                 *
            *           4          FIRST PASCAL TRACK                  *
            *           5          TRACK-TO-TRACK SKEW                 *
            *                                                          *
            *          NOTE: THE FORMAT PROGRAM TEMPORARILY (THAT IS,  *
            *                UNTIL ANOTHER COLD BOOT) ALTERS THE       *
            *                FORMAT OF DRIVES BY MODIFYING THESE       *
            *                INFORMATION BLOCKS. IF THE ADDRESS OF     *
            *                DSKINFO IS CHANGED, THE FORMAT PROGRAM    *
            *                MUST ALSO BE CHANGED.                     *
            *                                                          *
            ************************************************************

F4BB 4D001A0000DSKINFO DB   4DH,00H,1AH,00H,00H,01H,06H,00H,01H,00H,00H,00H    ;DRIVE A
F4C7 4D001A0000        DB   4DH,00H,1AH,00H,00H,01H,06H,00H,01H,00H,00H,00H    ;DRIVE B
F4D3 4D001A0000        DB   4DH,00H,1AH,00H,00H,01H,06H,00H,01H,00H,00H,00H    ;DRIVE C
F4DF 4D001A0000        DB   4DH,00H,1AH,00H,00H,01H,06H,00H,01H,00H,00H,00H    ;DRIVE D

F4EB 0000   JMPTBL  DB      00H,00H         ;STORAGE FOR POINTER TO JUMP TABLE

            ************************************************************
            *                                                          *
            * SETTRAK: ROUTINE TO SET CURRENT TRACK (CURTRAK).         *
            *                                                          *
            *          GIVEN  C = TRACK NUMBER                         *
            *                                                          *
            ************************************************************

F4ED 79     SETTRAK MOV     A,C
F4EE 3256F4         STA     CURTRAK
F4F1 C9             RET

            ************************************************************
            *                                                          *
            * SETSECT: ROUTINE TO SET CURRENT SECTOR (CURSECT).        *
            *                                                          *
            *          GIVEN  C = SECTOR NUMBER                        *
            *                                                          *
            ************************************************************

F4F2 79     SETSECT MOV     A,C
F4F3 3258F4         STA     CURSECT
F4F6 C9             RET

            ************************************************************
            *                                                          *
            * SETBUFR: ROUTINE TO SET CURRENT BUFFER ADDRESS (CURBUFR). *
            *                                                          *
            *          GIVEN  BC = BUFFER ADDRESS                      *
            *                                                          *
            ************************************************************

F4F7 215AF4 SETBUFR LXI     H,CURBUFR
F4FA 71             MOV     M,C
F4FB 23             INX     H
F4FC 70             MOV     M,B
F4FD C9             RET
```

CIRCLE 21 ON READER SERVICE CARD

INTELLIGENT VIDEO I/O FOR S-100 BUS



VIO-X

The VIO-X Video I/O Interface for the S-100 bus provides features equal to most intelligent terminals both efficiently and economically. It allows the use of standard keyboards and CRT monitors in conjunction with existing hardware and software. It will operate with no additional overhead in S-100 systems regardless of processor or system speed.

Through the use of the Intel 8275 CRT controller with an onboard 8085 processor and 4k memory, the VIO-X interface operates independently of the host system and communicates via two ports, thus eliminating the need for host memory space. The screen display rate is effectively 80,000 baud.

The VIO-X1 provides an 80 character by 25 line format (24 lines plus status line) using a 5 × 7 character set in a 7 × 10 dot matrix to display the full upper and lower case ASCII alphanumeric 96 printable character set (including true descenders) with 32 special characters for escape and control characters. An optional 2732 character generator is available which allows an alternate 7 × 10 contiguous graphics character set.

The VIO-X2 also offers an 80 character by 25 line format but uses a 7 × 7 character set in a 9 × 10 dot matrix allowing high-resolution characters to be used. This model also includes expanded firmware for block mode editing and light pen location. Contiguous graphics characters are not supported.

Both models support a full set of control characters and escape sequences, including controls for video attributes, cursor location and positioning, cursor toggle, and scroll speed. An onboard Real Time Clock (RTC) is displayed in the status line and may be read or set from the host system. A checksum test is performed on power-up on the firmware EPROM.

Video attributes provided by the 8275 in the VIO-X include:

- FLASH CHARACTER
- INVERSE CHARACTER
- UNDERLINE CHARACTER or
- ALT. CHARACTER SET
- DIM CHARACTER

The above functions may be toggled together or separately.

The board may be addressed at any port pair in the IEEE 696 (S-100) host system. Status and data ports may be swapped if necessary. Inputs are provided for parallel keyboard and for light pen as well as an output for audio signalling. The interrupt structure is completely compatible with Digital Research's MP/M ®.

Additional features include:

- HIGH SPEED OPERATION
- PORT MAPPED IEEE S-100 INTERFACE
- FORWARD/REVERSE SCROLL or
- PROTECTED SCREEN FIELDS
- CONVERSATIONAL or BLOCK MODE (opt)
- INTERRUPT OPERATION
- CUSTOM CHARACTER SET
- CONTROL CHARACTERS
- ESCAPE CHARACTER COMMANDS
- INTELLIGENT TERMINAL EMULATION
- TWO PAGE SCREEN MEMORY

VIO-X1 - 80 × 25 5 × 7 A & T **$295.00**
Conversational Mode
VIO-X2 - 80 × 25 7 × 7 A & T **$345.00**
Conversational & Block Modes

For European enquiries and sales contact:

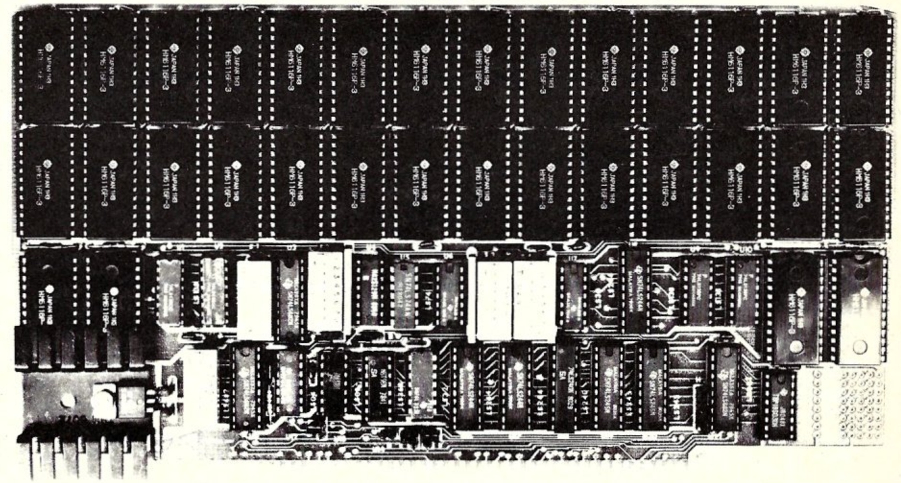
Fulcrum (Europe) England
Phone 061 828763

FULCRUM™
COMPUTER PRODUCTS

Distributed by:
WW COMPONENT SUPPLY INC. 1771 JUNCTION AVENUE • SAN JOSE, CA 95112 • (408) 295-7171

64K STATIC RAM BOARD FOR S-100 BUS
$425

NEW: Additional Features



FEATURES

- Conforms to IEEE 696 standard.
- 8 or 16 bit data transfers.
- 24 bit addressing.
- Bank select in 32K-32K or 48-16K.
- Banks selectable/deselectable on DMA.
- Responds to phantom pin 67 or 16.
- 2Kx8 static rams with 2716 pin out.
- Power consumption is typically 600 ma.
- Banks on or off on power up.
- Bank addressable to any of 256 possible ports.
- 8MHz with 150ns parts standard faster speeds available on request.
- Available partially loaded as a 32K board.
- Multiple bank residence.

NEW FEATURES

- Phantoms read only, or read and write.
- Generates onboard M-WRITE, or uses bus M-WRITE.
- Two seperate banks, independently addressable on 16K boundaries. Banks may be overlapped.

OMNIRAM REV. C

OMNIRAM INTRODUCTION:

The Fulcrum Computer Products OMNIRAM for the IEEE 696 (S100) bus provides 64 kilobytes of fast static random access memory. Provision is made for 8 or 16 bit transfers, extended 24 bit addressing, and for control via the bus phantom line. In addition, a number of features are included to make the OMNIRAM compatible with systems designed before the IEEE 696 standard was developed. These include bank selection and provision for operation with IMSAI-type front panels. When the bank select option is activated, the board is divided into two parts which can reside in separate banks. The division of the board may be into two 32K sections or into one 16K section and one 48K section. 2K blocks may be disabled in the upper 16K, or 4K blocks in the upper 32K, of memory. Provision is made for DMA overide of bank select if needed. The board is also compatible with IEEE 696 or IMSAI-type extended addressing.

Omniram 64 United States & Canada Prices Only

With Hitachi 6116	64K	32K
150ns. Rams	$470	$325
120ns. Rams	$550	$395
Board Without Ram	$200	
With Toshiba 2016 Rams	64K	32K
Typical current draw 800mill	$425	$275

FULCRUM™
COMPUTER PRODUCTS

Distributed by:
WW COMPONENT SUPPLY INC. 1771 JUNCTION AVENUE • SAN JOSE, CA 95112 • (408) 295-7171
For European enquiries and sales contact: **Fulcrum** (Europe) England. Phone 061 828763

```
                  ***********************************************************
                  *                                                         *
                  * DSKREAD: READ A SECTOR FROM A FLOPPY DISK AND REPORT A  *
                  *          STATUS FOR THE OPERATION.                      *
                  *                                                         *
                  *          RETURNS A = 0 IF OK                            *
                  *                    = 1 IF ERROR                         *
                  *                    = 9 IF OFF LINE                      *
                  *                                                         *
                  ***********************************************************

F4FE 3A54F4       DSKREAD LDA    CURDISK        ;READ CURRENT DRIVE NUMBER
F501 FE04                 CPI    4              ;LEGAL DRIVE NUMBER? (0-3)
F503 F236F5               JP     OFFLINE        ;REPORT ILLEGAL DRIVES AS OFF LINE
F506 4F                   MOV    C,A            ;STORE REQUESTED DRIVE NUMBER IN C
F507 CD1BF0               CALL   SELDRV         ;SELECT THE DRIVE
F50A 2156F4               LXI    H,CURTRAK
F50D 4E                   MOV    C,M
F50E CD0CF0               CALL   TRKSET         ;SET TRACK
F511 DA33F5               JC     READERR        ;ERROR?
F514 2158F4               LXI    H,CURSECT
F517 4E                   MOV    C,M
F518 CD0FF0               CALL   SETSEC         ;SET SECTOR
F51B DA33F5               JC     READERR        ;ERROR?
F51E 215AF4               LXI    H,CURBUFR
F521 4E                   MOV    C,M
F522 23                   INX    H
F523 46                   MOV    B,M
F524 CD12F0               CALL   SETDMA         ;SET READ ADDRESS
F527 DA33F5               JC     READERR        ;ERROR?
F52A CD15F0               CALL   DREAD          ;READ THE SECTOR
F52D D0                   RNC                   ;RETURN IF NO ERRORS DETECTED
F52E E680                 ANI    80H            ;CHECK FOR OFF LINE DISK
F530 C236F5               JNZ    OFFLINE
F533 3E01         READERR MVI    A,1
F535 C9                   RET                   ;RETURN 1 IF ERROR
F536 3E09         OFFLINE MVI    A,9
F538 C9                   RET                   ;RETURN 9 IF OFF LINE

                  ***********************************************************
                  *                                                         *
                  * DSKWRIT: WRITE A SECTOR TO A FLOPPY DISK AND REPORT A   *
                  *          STATUS FOR THE OPERATION.                      *
                  *                                                         *
                  *          RETURNS A = 0 IF OK                            *
                  *                    = 16 IF ERROR                        *
                  *                    = 9 IF OFF LINE                      *
                  *                                                         *
                  ***********************************************************

F539 3A54F4       DSKWRIT LDA    CURDISK        ;READ CURRENT DRIVE NUMBER
F53C FE04                 CPI    4              ;LEGAL DRIVE NUMBER? (0-3)
F53E F236F5               JP     OFFLINE        ;REPORT ILLEGAL DRIVES AS OFF LINE
F541 4F                   MOV    C,A            ;STORE REQUESTED DRIVE NUMBER IN C
F542 CD1BF0               CALL   SELDRV         ;SELECT THE DRIVE
F545 2156F4               LXI    H,CURTRAK
F548 4E                   MOV    C,M
F549 CD0CF0               CALL   TRKSET         ;SET TRACK
F54C DA6EF5               JC     WRITERR        ;ERROR?
F54F 2158F4               LXI    H,CURSECT
F552 4E                   MOV    C,M
F553 CD0FF0               CALL   SETSEC         ;SET SECTOR
F556 DA6EF5               JC     WRITERR        ;ERROR?
F559 215AF4               LXI    H,CURBUFR
F55C 4E                   MOV    C,M
F55D 23                   INX    H
F55E 46                   MOV    B,M
F55F CD12F0               CALL   SETDMA         ;SET READ ADDRESS
F562 DA6EF5               JC     WRITERR        ;ERROR?
F565 CD18F0               CALL   DWRITE         ;WRITE THE SECTOR
F568 D0                   RNC                   ;RETURN IF NO ERRORS DETECTED
F569 E680                 ANI    80H            ;CHECK FOR OFF LINE DISK
F56B C236F5               JNZ    OFFLINE
F56E 3E10         WRITERR MVI    A,16
F570 C9                   RET                   ;RETURN 16 IF ERROR

                  ***********************************************************
                  *                                                         *
                  * DSKINIT: RESET A DISK AND REPORT STATUS FOR THE OPERATION *
                  *                                                         *
                  *          RETURNS A = 0 IF OK                            *
                  *                    = 9 IF OFF LINE                      *
                  *                                                         *
                  ***********************************************************

F571 AF           DSKINIT XRA    A
F572 C9                   RET

                  ***********************************************************
                  *                                                         *
                  * DSKSTRT: ACTIVATE THE CURRENT DISK.                     *
                  *                                                         *
                  ***********************************************************

F573 C9           DSKSTRT RET

                  ***********************************************************
                  *                                                         *
                  * DSKSTOP: DEACTIVATE THE CURRENT DISK.                   *
                  *                                                         *
                  ***********************************************************

F574 C9           DSKSTOP RET

                  ***********************************************************
                  *                                                         *
                  * PRNINIT: INITIALIZE PRINTER AND REPORT STATUS OF THE    *
                  *          CONNECTION.                                    *
                  *                                                         *
                  *          RETURNS  A = 0 IF PRINTER ON LINE              *
                  *                     = 9 IF PRINTER OFF LINE             *
                  *                                                         *
                  ***********************************************************

F575 AF           PRNINIT XRA    A              ;REPORT PRINTER ON LINE, EVEN
                                                ;  IF IT IS NOT, SINCE OTHERWISE
                                                ;  MUST REINITIALIZE PASCAL TO
                                                ;  GET IT TO REALIZE PRINTER HAS
                                                ;  BEEN TURNED ON.  THIS STATUS IS
                                                ;  ONLY CHECKED AT INITIALIZATION.

F576 C9                   RET

                  ***********************************************************
                  *                                                         *
                  * PRNSTAT: REPORT STATUS OF PRINTER CONNECTION AND PRINTER *
                  *          INPUT CHANNEL.                                 *
                  *                                                         *
                  *          RETURNS  A = 0 IF PRINTER ON LINE              *
                  *                     = 9 IF PRINTER OFF LINE             *
                  *                   C = 0 IF NO CHAR                      *
                  *                     = FF IF CHARACTER AVAILABLE         *
                  *                                                         *
                  ***********************************************************
```

```
F577 AF         PRNSTAT XRA     A               ;ZERO A REGISTER TO INDICATE PRINTER
                                                ;  ON LINE, EVEN IF IT IS NOT, SINCE
                                                ;  CANNOT READ FROM THIS PRINTER
                                                ;  AND STATUS APPLIES TO READ ONLY
F578 0E00               MVI     C,0             ;REPORT NO CHARACTER READY
F57A C9                 RET

                ************************************************************
                *                                                          *
                * PRNREAD: RECEIVE A CHARACTER FROM THE PRINTER.           *
                *                                                          *
                *         RETURNS  A = 0 IF OK                             *
                *                    = 1 IF ERROR                          *
                *                    = 9 IF OFF LINE                       *
                *                  C = INPUT CHARACTER                     *
                *                                                          *
                ************************************************************

F57B AF         PRNREAD XRA     A               ;REPORT PRINTER ON LINE
F57C C9                 RET

                ************************************************************
                *                                                          *
                * PRNWRIT: WRITE A CHARACTER TO THE PRINTER AND REPORT     *
                *          STATUS OF THE CONNECTION.                       *
                *                                                          *
                *          GIVEN    C = OUTPUT CHARACTER                   *
                *          RETURNS  A = 0 IF OK                            *
                *                     = 9 IF OFF LINE                      *
                *                                                          *
                ************************************************************

F57D 21C0F5     PRNWRIT LXI     H,CHCNT         ;HL POINTS TO COLUMN COUNTER
F580 79                 MOV     A,C             ;GET CHARACTER
F581 FE0D               CPI     CR              ;IS IT A CARRIAGE RETURN?
F583 CA9FF5             JZ      CRROUT          ;YES, PROCESS IT
F586 FE09               CPI     HT              ;IS IT A TAB?
F588 CAA4F5             JZ      TAB             ;YES, PROCESS IT
F58B FE20               CPI     SPC             ;IS IT A NON-PRINTING CHAR?
F58D FA91F5             JM      PRNOUT          ;IF LESS THAN A SPACE, SKIP COUNTING
F590 34                 DB      34H             ;INC  (HL) - INCREMENT COLUMN COUNTER

F591 DB84       PRNOUT: IN      BPARPT          ;GET STATUS OF PAPER TIGER
F593 E605               ANI     5H              ;WAIT UNTIL OK TO SEND
F595 FE04               CPI     4H              ;WAIT FOR FAULT HIGH AND BUSY LOW
F597 C291F5             JNZ     PRNOUT          ;
F59A 79                 MOV     A,C             ;OUTPUT THE CHARACTER
F59B D384               OUT     BPARPT          ;
F59D AF                 XRA     A               ;ZERO A REG. TO REPORT PTR ON LINE
F59E C9                 RET

F59F AF         CRROUT: XRA     A               ;<CR> RECEIVED, SO ZERO THE COUNTER
F5A0 77                 DB      77H             ;LD  (HL),A
F5A1 C391F5             JMP     PRNOUT

F5A4            TAB:    ORG     $               ;REPLACE <HT> WITH MULTIPLE SPACES
F5A4 3AC0F5             LDA     CHCNT           ;LOAD COLUMN POSITION
F5A7 E607               ANI     7H              ;MODULO 8
F5A9 DE08               SBI     8               ;CALC. MINUS DIST. TO NEXT TAB STOP
F5AB 47                 MOV     B,A             ;SAVE DISTANCE IN B
F5AC DB84       SPACES  IN      BPARPT          ;GET STATUS OF PAPER TIGER
F5AE E605               ANI     5H              ;MASK OUT ALL BUT BUSY AND FAULT BITS
F5B0 FE04               CPI     4H              ;WAIT FOR BUSY LOW AND FAULT HIGH
F5B2 C2ACF5             JNZ     SPACES          ;LOOP UNTIL READY FOR OUTPUT
F5B5 3E20               MVI     A,SPC           ;LOAD A SPACE INTO REGISTER A
F5B7 D384               OUT     BPARPT          ;OUTPUT IT
F5B9 34                 DB      34H             ;INC  (HL)  - INCREMENT COLUMN COUNT
```

```
F5BA 04                 INR     B               ;INCREMENT COUNTER
F5BB FAACF5             JM      SPACES          ;LOOP UNTIL COUNTER = 0
F5BE AF                 XRA     A               ;REPORT PRINTER ON LINE
F5BF C9                 RET

F5C0 00         CHCNT:  DB 00

                ************************************************************
                ************************************************************
                *
                *       I/O  DRIVER FOR THE DTC 300/S
                *
                *       TO USE THE DTC AS THE PRINTER, REPLACE THE PRECEDING
                *       PRNINIT, PRNSTAT, PRNREAD, AND PRNWRIT ROUTINES WITH
                *       THIS CODE. THE DTC SHOULD BE CONNECTED TO THE SERIAL B PORT.
                *
                ************************************************************
                *
                *
                *PRNINIT        MVI     A,BR300
                *       OUT     BSTATPT
                *       XRA     A
                *       RET
                *
                *
                *PRNSTAT        XRA     A
                *       MVI     C,0
                *       RET
                *
                *
                *PRNREAD        XRA     A
                *       RET
                *
                *
                *PRNWRIT        ORG     $               ;LINE PRINTER (DIABLO)
                *       MOV     A,C             ;GET CHARACTER
                *       CPI     CR              ;IS IT A CARRIAGE RETURN?
                *       JZ      CRROUT          ;YES, PROCESS IT
                *       CPI     HT              ;IS IT A TAB?
                *       JZ      TAB             ;YES, PROCESS IT
                *       CPI     ESC             ;IS IT AN ESCAPE?
                *       JZ      DECR            ;DECREMENT CHARCOUNT
                *       CPI     BS              ;IS IT A BACKSPACE?
                *       JZ      DECR            ;DECREMENT CHARCOUNT
                *       CPI     SPC             ;IS IT A NON-PRINTING CHAR?
                *       JM      LIST1           ;IF LESS THAN A SPACE, SKIP COUNTING
                *       LDA     CHCNT           ;GET THE CHARCOUNT FOR THE LINE
                *       INR     A
                *       STA     CHCNT           ;INCREMENT COUNT & STORE
                *       MOV     A,C
                *       CPI     SPC             ;IS IT A SPACE?
                *       JZ      SPACEROUT
                *       LDA     SPCNT
                *       ANA     A               ;GET THE CPU FLAGS SET
                *       JZ      LIST1
                *       PUSH    B               ;SAVE THE CHARACTER
                *       CPI     3               ;3 SPACES?
                *       JNC     ESCSEQ          ;IF MORE THAN 3 SPACES, DO ESC SEQ
                *       MVI     C,SPC           ;OUTPUT THE SPACES
                *       MOV     B,A             ;LOAD COUNTER
                *SPAGAIN:CALL   LIST1           ;IF 3 SPACES OR LESS, PRINT THEM
                *       DCR     B
                *       JNZ     SPAGAIN
                *       JMP     NCHAR1
                *ESCSEQ:        MVI     C,ESC           ;DIRECT HORIZONTAL TAB
                *       CALL    LIST1           ;SEQUENCE IS <ESC> T <CHARACTER>
                *       MVI     C,'T'
```

```
*       CALL    LIST1
*       LDA     CHCNT
*       DCR     A
*       MOV     C,A
*       CALL    LIST1
*NCHAR1:        ORG     $               ;RESET SPACE COUNT
*       XRA     A
*       STA     SPCNT
*       POP     B
*       JMP     LIST1
*
*SPACEROUT:     ORG $           ;INCREMENT SPACE COUNT
*       LDA     SPCNT
*       INR     A
*       STA     SPCNT
*       XRA     A               ;CLEAR AC FOR PASCAL
*       RET
*
*CRROUT:        ORG     $
*       XRA     A
*       STA     SPCNT           ;ZERO OUT SPACE COUNT TOO
*       STA     CHCNT           ;ZERO OUT THE COUNT
*       JMP     LIST1
*
*TAB:   ORG     $               ;DIRECT HORIZONTAL TAB TO REPLACE <HT>
*       LDA     CHCNT           ;LOAD COLUMN POSITION
*LOOP:  SBI     8               ;FIND DISTANCE TO NEXT TAB STOP
*       JP      LOOP            ;USING REPEATED SUBTRACTION
*       MOV     B,A
*       LDA     CHCNT
*       SUB     B
*       STA     CHCNT           ;STORE NEW POSITION
*       LDA     SPCNT           ;INCREMENT SPCNT TO NEXT TAB STOP
*       SUB     B               ;
*       STA     SPCNT           ;STORE NEW SPCNT
*       XRA     A               ;CLEAR AC FOR PASCAL
*       RET                     ;RETURN
*
*DECR:  LDA CHCNT               ;DECREMENT COLUMN COUNT
*       DCR A
*       STA CHCNT               ;STORE IT
*       JMP LIST1
*
*CHCNT: DB 00
*SPCNT: DB 00
*
*
*LIST1: IN      BSTATPT         ;GET THE STATUS
*       ANI     TXRDY           ;MASK THE TRANSMIT BUFFER EMPTY BIT
*       JZ      LIST1           ;LOOP UNTIL READY TO OUTPUT
*       MOV     A,C             ;GET THE DATA TO OUTPUT
*       OUT     BSERPT          ;OUTPUT IT
*       XRA     A               ;CLEAR AC FOR PASCAL
*       RET
*
*
***************************************************************
***************************************************************

*************************************************************
*                                                           *
* REMINIT: INITIALIZE THE REMOTE PORT AND REPORT STATUS     *
*          OF THE CONNECTION.                               *
*                                                           *
*          RETURNS  A = 0 IF REMOTE ON LINE                 *
*                     = 9 IF REMOTE OFF LINE                *
*                                                           *
*************************************************************

F5C1 AF         REMINIT XRA     A
F5C2 C9                 RET

        ************************************************************
        *                                                          *
        * REMSTAT: REPORT STATUS OF REMOTE CONNECTION AND REMOTE   *
        *          INPUT CHANNEL.                                  *
        *                                                          *
        *          RETURNS  A = 0 IF REMOTE ON LINE                *
        *                     = 9 IF REMOTE OFF LINE               *
        *                   C = 0 IF NO CHARACTER                  *
        *                     = FF IF CHARACTER AVAILABLE          *
        *                                                          *
        ************************************************************

F5C3 CD21F0     REMSTAT CALL    TSTAT
F5C6 AF                 XRA     A
F5C7 0EFF               MVI     C,0FFH
F5C9 C8                 RZ
F5CA 0E00               MVI     C,0
F5CC C9                 RET

        ************************************************************
        *                                                          *
        * REMREAD: RECEIVE A CHARACTER FROM THE REMOTE INPUT       *
        *          CHANNEL AND REPORT THE STATUS OF THE REMOTE     *
        *          CONNECTION.                                     *
        *                                                          *
        *          RETURNS  A = 0 IF OK                            *
        *                     = 1 IF ERROR                         *
        *                     = 9 IF OFF LINE                      *
        *                   C = INPUT CHARACTER                    *
        *                                                          *
        ************************************************************

F5CD CD03F0     REMREAD CALL    REMIN
F5D0 4F                 MOV     C,A
F5D1 AF                 XRA     A
F5D2 C9                 RET

        ************************************************************
        *                                                          *
        * REMWRIT: WRITE A CHARACTER TO THE REMOTE PORT AND RETURN *
        *          THE STATUS OF THE CONNECTION.                   *
        *                                                          *
        *          GIVEN    C = OUTPUT CHARACTER                   *
        *          RETURNS  A = 0 IF OK                            *
        *                     = 9 IF OFF LINE                      *
        *                                                          *
        ************************************************************

F5D3 CD06F0     REMWRIT CALL    REMOUT
F5D6 AF                 XRA     A
F5D7 C9                 RET

        ************************************************************
        *                                                          *
        * USRINIT: INITIALIZE A USER-DEFINED DEVICE AND RETURN THE *
        *          STATUS OF THE DEVICE.                           *
        *                                                          *
        *          GIVEN    C = DEVICE NUMBER                      *
        *          RETURNS  A = 0 IF DEVICE ON LINE                *
        *                     = 9 IF DEVICE OFF LINE               *
        *                                                          *
        ************************************************************

F5D8 3E09       USRINIT MVI     A,9
F5DA C9                 RET
```

```
                ************************************************************
                *                                                          *
                * USRSTAT: REPORT STATUS OF A USER-DEFINED DEVICE.         *
                *                                                          *
                *         GIVEN   SP = RETURN ADDRESS                      *
                *                    = I/O DIRECTION                       *
                *                    = STATUS RECORD NUMBER                *
                *                    = DEVICE NUMBER                       *
                *         RETURNS  A = 0 IF DEVICE ON LINE                 *
                *                    = 9 IF DEVICE OFF LINE                *
                *                                                          *
                ************************************************************

F5DB E1         USRSTAT POP     H               ;GET RETURN ADDRESS
F5DC C1                 POP     B               ;CLEAN UP STACK
F5DD C1                 POP     B
F5DE C1                 POP     B
F5DF 3E09               MVI     A,9             ;UNIMPLEMENTED, SO REPORT AS OFF LINE
F5E1 E9                 PCHL                    ;RETURN

                ************************************************************
                *                                                          *
                * USRREAD: GET STATUS OF A USER-DEFINED DEVICE, AND PERFORM *
                *          A READ OPERATION ON IT.                         *
                *                                                          *
                *         GIVEN   SP = RETURN ADDRESS                      *
                *                    = EXTRA PARAMETER 2                   *
                *                    = EXTRA PARAMETER 1                   *
                *                    = POINTER TO BUFFER                   *
                *                    = DEVICE NUMBER                       *
                *                    = EXTRA PARAMETER 3                   *
                *         RETURNS  A = 0 IF DEVICE ON LINE                 *
                *                    = 9 IF DEVICE OFF LINE                *
                *                                                          *
                ************************************************************

F5E2 E1         USRREAD POP     H               ;GET RETURN ADDRESS
F5E3 C1                 POP     B               ;CLEAN UP STACK
F5E4 C1                 POP     B
F5E5 C1                 POP     B
F5E6 C1                 POP     B
F5E7 C1                 POP     B
F5E8 3E09               MVI     A,9             ;UNIMPLEMENTED - REPORT AS OFF LINE
F5EA E9                 PCHL                    ;RETURN

                ************************************************************
                *                                                          *
                * USRWRIT: GET STATUS OF A USER-DEFINED DEVICE, AND PERFORM *
                *          A WRITE OPERATION ON IT.                        *
                *                                                          *
                *         GIVEN   SP = RETURN ADDRESS                      *
                *                    = EXTRA PARAMETER 2                   *
                *                    = EXTRA PARAMETER 1                   *
                *                    = POINTER TO BUFFER                   *
                *                    = DEVICE NUMBER                       *
                *                    = EXTRA PARAMETER 3                   *
                *         RETURNS  A = 0 IF DEVICE ON LINE                 *
                *                    = 9 IF DEVICE OFF LINE                *
                *                                                          *
                ************************************************************

F5EB E1         USRWRIT POP     H               ;GET RETURN ADDRESS
F5EC C1                 POP     B               ;CLEAN UP STACK
F5ED C1                 POP     B
F5EE C1                 POP     B
F5EF C1                 POP     B
F5F0 C1                 POP     B
F5F1 3E09               MVI     A,9             ;UNIMPLEMENTED - REPORT AS OFF LINE
F5F3 E9                 PCHL                    ;RETURN

                ************************************************************
                *                                                          *
                * CLKREAD: REPORT STATUS OF THE REAL-TIME CLOCK, AND RETURN *
                *          THE CURRENT TIME.                               *
                *                                                          *
                *          RETURNS  A = 0 IF CLOCK ON LINE                 *
                *                     = 9 IF CLOCK OFF LINE                *
                *                    DE = LEAST SIGNIFICANT WORD           *
                *                    HL = MOST SIGNIFICANT WORD            *
                *                                                          *
                ************************************************************

F5F4 3E09       CLKREAD MVI     A,9             ;UNIMPLEMENTED - REPORT AS OFF LINE
F5F6 C9                 RET
```

Listing 3

```
(***********************************************************************)
(*                                                                     *)
(*      THIS PROGRAM REFORMATS DISK DRIVES RUNNING UNDER UCSD          *)
(*      PASCAL BY MODIFYING THE DISK INFORMATION BLOCK                 *)
(*      (DSKINFO) WITHIN THE SBIOS. IF THE SBIOS IS ALTERED,           *)
(*      CHANGING THE ADDRESS OF DSKINFO, THEN THE EQUATE               *)
(*      WITHIN THE REFORMAT SUBROUTINE MUST BE CHANGED AS WELL.        *)
(*      REFORMAT IS LOCATED INSIDE THE SYSTEM LIBRARY.                 *)
(*                                                                     *)
(***********************************************************************)

PROGRAM FORMAT;

VAR
 DENSITY:INTEGER;                       (* BYTES PER SECTOR *)
 INTERLEAVING:INTEGER;                  (* INTERLEAVING RATIO *)
 FIRST_PASCAL_TRACK:INTEGER;            (* FIRST TRACK USED BY PASCAL *)
 SKEW:INTEGER;                          (* TRACK-TO-TRACK SKEW *)
 OFFSET:INTEGER;                        (* OFFSET INTO SBIOS DSKINFO BLOCK *)
 DRIVE:INTEGER;                         (* NUMBER OF FLOPPY DISK *)
 CH:CHAR;

PROCEDURE REFORMAT(PARAM1,PARAM2,PARAM3,PARAM4,OFFSET:INTEGER); EXTERNAL;

BEGIN
 WRITELN;
 WRITELN('EMU floppy disk formatter [1.0]');
 WRITE('Reformat which drive (A,B,C, or D)? ');  READLN(CH);
  CASE CH OF
   'A','a':     DRIVE:=0;
   'B','b':     DRIVE:=1;
   'C','c':     DRIVE:=2;
   'D','d':     DRIVE:=3;
  END;
  IF CH IN ['A','a','B','b','C','c','D','d']
    THEN
      BEGIN
       OFFSET := 12*DRIVE+4;
       WRITELN;
       WRITE('Track-to-track skew: ');         READLN(SKEW);
       WRITE('Interleaving factor: ');         READLN(INTERLEAVING);
       WRITE('Bytes per sector: ');            READLN(DENSITY);
       WRITE('First Pascal track: ');          READLN(FIRST_PASCAL_TRACK);
```

```
;***********************************************************************
;                                                                      *
;   NAME:      REFORMAT                                                *
;                                                                      *
;   PURPOSE:   ALTER THE RECORDING FORMAT OF DISK DRIVES BY MODIFYING  *
;              THE DISK INFORMATION BLOCK (DSKINFO) INSIDE THE SBIOS.  *
;              THIS ROUTINE IS USED BY THE FORMAT PROGRAM.             *
;                                                                      *
;   USE:       REFORMAT(SKEW,FIRST_TRACK,INTERLEAVING,BYTES,OFFSET)    *
;                                                                      *
;              WHERE   SKEW          =  TRACK-TO-TRACK SKEW            *
;                      FIRST_TRACK   =  FIRST PASCAL TRACK             *
;                      INTERLEAVING  =  INTERLEAVING RATIO             *
;                      BYTES         =  NUMBER OF BYTE PER SECTOR      *
;                      OFFSET        =  OFFSET INTO DSKINFO            *
;                                                                      *
;***********************************************************************

.PROC           REFORMAT,5

DSKINFO .EQU    0F4BBH          ;BEGINING OF DISK INFORMATION BLOCK IN SBIOS

        POP     IX              ;GET RETURN ADDRESS
        POP     HL              ;GET OFFSET FOR DSKINFO
        LD      DE,DSKINFO      ;GET STARTING ADDRESS OF DSKINFO
        ADD     HL,DE           ;COMPUTE STARTING ADDRESS + OFFSET
        EX      DE,HL           ;DE = DESTINATION FOR LDIR TRANSFER
        LD      BC,8            ;BYTE COUNT IN BC (MOVE 4 WORDS=8 BYTES)
        LD      HL,00           ;HL <-- SP
        ADD     HL,SP           ;USE HL AS SOURCE FOR LDIR TRANSFER
        LDIR                    ;TRANSFER PARAMETERS DIRECTLY FROM
                                ;STACK TO DSKINFO BLOCK
        LD      SP,HL           ;RESET STACK POINTER
        JP      (IX)            ;RETURN TO PASCAL
        .END
```

Listing 4

```
***************************************************************
*                                                             *
*                          BOOTMKR                            *
*                                                             *
*   THIS PROGRAM IS USED FOR MODIFICATION OF THE UCSD PAS-    *
*  CAL PRIMARY BOOTSTRAP LOADER (PRIMARY) AND/OR EXTENDED     *
*  SBIOS (SBIOS7).  THESE PROGRAMS CAN BE MODIFIED USING THE  *
*  CP/M EDITOR AND REASSEMBLED UNDER CP/M.  THEN DDT MUST BE  *
*  USED TO READ THE NEW .HEX FILES INTO MEMORY.  THE PRIMARY  *
*  BOOTSTRAP MUST BE READ WITH AN APPROPRIATE OFFSET SO THAT  *
*  IT BEGINS AT 980H, AND THE SBIOS SO THAT IT BEGINS AT A00H. *
*  THEN BOOTMKR WILL COPY THE TWO PROGRAMS ONTO THE PROPER    *
*  SECTORS OF TRACK ZERO OF A UCSD PASCAL DISKETTE.  SINCE    *
*  THE SECONDARY BOOTSTRAP WILL NOT BE COPIED, WHENEVER A NEW *
*  DISKETTE IS TO BE LOADED WITH A NEW PRIMARY BOOTSTRAP AND  *
*  SBIOS, THE UCSD PASCAL PROGRAM BOOTER SHOULD BE USED FIRST *
*  TO COPY ALL OF TRACK ZERO FROM AN EXISTING UCSD PASCAL     *
*  DISKETTE.                                                  *
*                                                             *
***************************************************************
```

```
                    *****************************************
                    *                                       *
                    *   EQUATES FOR DISK JOCKEY 2D          *
                    *                                       *
                    *****************************************

F000 =              ORIGIN  EQU     0F000H          ;DJ/2D firmware starting address
F01B =              SELDRV  EQU     ORIGIN+27       ;Disk drive selection routine
F009 =              TKZERO  EQU     ORIGIN+9        ;Track zero seek routine
F012 =              SETDMA  EQU     ORIGIN+18       ;Set read/write begin address in mem.
F00F =              SETSEC  EQU     ORIGIN+15       ;Select sector routine
F024 =              DMAST   EQU     ORIGIN+36       ;Read current read/write address
F018 =              DWRITE  EQU     ORIGIN+24       ;Disk write routine

0005 =              ENTRY   EQU     5H              ;Entry point for BDOS
000D =              CR      EQU     0DH             ;Carriage return
000A =              LF      EQU     0AH             ;Line feed
    WRITELN;
    WRITELN('Type "R" to reformat drive ',ch,'.');
    WRITE('Any other character exits with no changes. ');  READLN(CH);
    IF CH IN ['R','r']
      THEN BEGIN
        REFORMAT(SKEW,FIRST_PASCAL_TRACK,INTERLEAVING,DENSITY,OFFSET);
        WRITELN('Reformatting complete');  END
      ELSE WRITELN('No changes made.');
  END;
WRITELN;
WRITELN('Returning to Pascal')
 END.
```

UCSD Adaptable Pascal Versions II.0 and IV.0 *continued . . .*

```
0100                    ORG     100H

0100 319402             LXI     SP,STACK
0103 116F01             LXI     D,PROMPT1       ;Send first prompt
0106 CD6901             CALL    PBUFF           ; to console
0109 CD6301             CALL    RDCON           ;Get character from console
010C 319402     START:  LXI     SP,STACK
010F AF                 XRA     A               ;Clear flags
0110 0E00               MVI     C,0
0112 CD1BF0             CALL    SELDRV          ;Select drive no. zero, i.e. drive A
0115 CD09F0             CALL    TKZERO          ;Select track zero
0118 018009             LXI     B,980H          ;Prepare to read memory
011B CD12F0             CALL    SETDMA          ; beginning at 980H
011E 0E01               MVI     C,1             ;Select sector no. 1
0120 CD0FF0             CALL    SETSEC
0123 CD18F0             CALL    DWRITE          ;Write to disk
0126 DA0C01             JC      START           ;Carry set indicates write error

0129 011308     WRITE:  LXI     B,813H          ;SBIOS goes on sectors 19 to 26
                                                ; i.e. 13H to 1AH
012C C5                 PUSH    B               ;Use B register as counter
012D 01000A             LXI     B,0A00H         ;Prepare to read SBIOS
0130 CD12F0     WRLOOP  CALL    SETDMA          ;Set memory address
0133 C1                 POP     B               ;Get proper sector no. in B
0134 C5                 PUSH    B
0135 CD0FF0             CALL    SETSEC
0138 CD18F0             CALL    DWRITE
013B DA2901             JC      WRITE           ;Repeat if error on write
013E C1                 POP     B
013F 05                 DCR     B               ;Decrement sector counter
0140 CA5101             JZ      DONE            ;Exit when all 8 sectors written
0143 0C                 INR     C
0144 C5                 PUSH    B               ;Increment sector no. to write
0145 CD24F0             CALL    DMAST           ;Find current memory address
0148 218000             LXI     H,80H
014B 09                 DAD     B               ;Add 128 (80H) to mem. address
014C E5                 PUSH    H
014D C1                 POP     B               ;Get new mem. address in B
014E C33001             JMP     WRLOOP
0151 114702     DONE:   LXI     D,PROMPT2       ;Second prompt to console
0154 CD6901             CALL    PBUFF
0157 CD6301             CALL    RDCON           ;Get character from console
015A AF                 XRA     A
015B 0E00               MVI     C,0             ;Select drive zero (A)
015D CD1BF0             CALL    SELDRV
0160 C30000             JMP     00H             ;Warm boot

0163 0E01       RDCON   MVI     C,1             ;Use CP/M function 'read console'
0165 CD0500             CALL    ENTRY           ; to get character
0168 C9                 RET

0169 0E09       PBUFF   MVI     C,9             ;Use CP/M function 'print console
016B CD0500             CALL    ENTRY           ; buffer' to output message
016E C9                 RET

016F 0D0A424F4FPROMPT1 DB       CR,LF,'BOOTSTRAP MODIFIER FOR UCSD PASCAL [1.1]',CR,LF
019B 0D0A4E4F54         DB      CR,LF,'NOTE: BEFORE USING THIS PROGRAM, THE PRIMARY'
01C9 0D0A424F4F         DB      CR,LF,'BOOTSTRAP AND CBIOS MUST BE LOADED INTO'
01F2 0D0A4D454D         DB      CR,LF,'MEMORY AT 980H AND A00H, RESPECTIVELY.'
021A 0D0A0A504C         DB      CR,LF,LF,'PLACE PASCAL DISK IN DRIVE A, THEN RETURN$'
0247 0D0A494E53PROMPT2 DB       CR,LF,'INSERT SYSTEM DISK IN DRIVE A, THEN RETURN','$'
0274                    DS      20H
0294 =          STACK   EQU     $
```

Saving Program State under UCSD Pascal

A description of procedures that aid manipulation of list data

by Jon Bondy

I was recently contacted by a friend of mine (Herman Euwema of Scholar Computer, Princeton, NJ) who had an unusual UCSD Pascal problem to be solved. He had written a Lisp interpreter which he was using to manipulate dynamic list data structures (what else?!). Although part of his program provided a way to save and restore list data in text form, he wanted a way to save his program "state" more rapidly, even if only temporarily. This problem is not a simple one, since it requires saving both static and dynamic data structures. Intrigued, I started thinking about the problem, and one thing led to another. . . .

When one runs a Pascal program, the data in the program which determine the current program "state" (or the "memory" of what the program is doing at a particular time, if you wish) resides in two principle areas of memory: that portion of the stack devoted to storing "global" variables, and that portion of memory in which the "heap" is stored. Just in case you are not completely familiar with these data storage approaches, I will discuss each of them briefly.

Global Variable Storage

Although programs may contain variables declared at all procedure levels, those declared at the outermost or program level are of particular interest for two reasons. First, they are visible through all of the program, and are therefore sometimes referred to as "global" variables. Second, in contrast to "local" variables declared within procedures, global variables exist throughout the entire program execution time; they are allocated "statically."

By the definition of the Pascal language, local variables are temporary; storage for them is allocated (of a stack) when the procedure is called, and that storage is deallocated when the procedure finishes execution. This means that one cannot store a value in a local variable during one invocation of a procedure and expect that it will still be there when the procedure is invoked again. One immediate reaction to this might be "But that is absurd: why was Pascal designed so carelessly?" In fact, it is exactly this kind of behaviour which allows all Pascal procedures to be "recursive": In providing the powerful tool of recursion, Pascal denies the programmer static local variables. Of course, Pascal could have provided static local variables as well as dynamic ones, as Algol and C do, but that is another discussion altogether.

The fact that all local variables are non-static means that only the global variables can contain true program state information; the local variables are too ephemeral to contain anything except temporary state information. Because of this, we need only save and restore global variables when preserving program state.

The Heap

The "Heap" is a portion of memory dedicated to "dynamic storage" of variables. There is a sense in which the stack is used dynamically, in that each time a procedure is invoked a portion of the stack is used to store the procedure's parameters, local variables, and temporary storage (expression evaluation, etc.); the Heap is dynamic in a different way. The difference is that the stack expands and contracts in a regular and predictable fashion: procedure calls use stack space and procedure exits return stack space. The Heap expands when the programer explicitly requests memory allocation (via the intrinsic NEW), and does not contract until the storage is released (via the intrinsic DISPOSE, if this is implemented. It is not implemented in the different versions of UCSD Pascal prior to IV.0).

By their very nature, variables which are stored

Jon Bondy, Box 148, Ardmore PA 19003

on the heap are not named in the standard fashion. A standard variable always exists within its lexical scope (i.e., if you can use its name in a program, it will exist at that point in the program's execution); a dynamic variable is certain not to exist during some portion of program execution. In order to access these variables, Pascal uses "pointers." A pointer is a variable which "points to" or "contains the address of" (for you assembly language addicts) a dynamic variable; the dynamic variable may not have a name, but it may be accessed via the pointer, which does have a name.

Pointers can point only to variables of a single declared type, allowing Pascal to preserve type checking when operating on either dynamic or static variables. If a pointer "P" is to point to an integer variable, we declare TYPE P : INTEGER; "P" points to an integer. When one wants to create a dynamic variable, one calls NEW passing to it a pointer variable; NEW allocates the storage and sets the pointer variable pointing to the new dynamic variable. One can then refer to the new dynamic variable by using the pointer variable and the "↑" symbol. If one said NEW(P), then one could increment that integer with "P↑ := P↑ + 1;".

This may seem needlessly complex, but the principle beauty of dynamic data structures lies in the cases where the allocated variables are records, and their fields themselves contain pointers. If we declare a record like

```
list_ptr = ↑list_rec;
list_rec = record
   data : integer;
   next : list_ptr;
   end;
```

then if one had a "list_ptr" variable called "root," one could create a "linked list" of such records as follows:

```
NEW(root);
NEW(root↑.next);
NEW(root↑.next↑.next);
```

These kinds of data structure (and more complex ones with more than one link in the record) hold two kinds of beauty (utility?). First, they can grow to fill all available memory without deciding in advance how large they must or will become; one never has arrays which are too large or too small. Second, extremely complex structures of data can be created by utilizing multiple link fields. A link between two records really makes them "logically adjacent," no matter where they lie in memory; complex "next to" relationships can be described using links in complex ways.

Note that each dynamic data structure is "rooted" in the global static variables: to get to an element in the linked list, we start at the root and walk the chain of dynamic records. It is this relationship which allows us to save an entire program state properly: the global variables point to the dynamic data, and (in some programs, at least) the dynamic data contains the real program state.

Languages like Lisp (LISt Processor) use the heap areas heavily to create linked lists and trees of data and manipulate them. Transposing a linear structure to/from a Text file from/to the heap is fairly straightforward (although time consuming), but attempting to transfer a tree-like structure to/from a Text file from/to the heap is more complex. When attempting to write a (two-dimensional) tree structure to a (one-dimensional) file, at the first branch one is confronted with the question "Does the right branch or the left branch go into the file next? Where does the other branch go?" The topography of complex linked data structures is such that file generation is much more difficult and time consuming. As an example, the Lisp processor which my friend was using could take as long as 15 minutes compute-bound to write the Lisp data structures to disk; this was on a WD MicroEngine, too—a processor which is about 8 times faster than a Z80 at executing UCSD Pascal.

Clearly, the best way to deal with the problem would be to save both the static (global variables) and dynamic (heap) data structures on disk and read them back in when required. Although this sounds simple, this approach needs a little care in order that it work properly. Most Pascal implementations store pointer values as hardware addresses; this is in contrast to integer or real values, which have nothing to do with physical locations in the hardware. If one were to attempt to restore data from disk and discover that it was not placed in EXACTLY the same locations as it was originally, this would cause immense problems. Consider our linked list above: the "root" variable would point to what it thought was a record, but if the record was not in the correct location, garbage would be processed instead. In order to ensure correct processing, one must make NO changes in either the program or the operating system between

. . . the current program "state" resides in "global" variables and in the "Heap."

saving and restoring the data; even a change of one byte will cause the technique to crash the system. If you decide to employ this technique, be aware that use of different versions of the same operating system, BIOS (i.e. loadable drivers), or program between saves and restores could cause system crashes.

Writing Global Variables to Disk

Writing the global variables to disk would be fairly easy if it were not for a few complicating factors. First, since the variables are placed on the stack in the order in which they are declared, they are in the REVERSE order in memory, as the stack grows DOWN. This means that attempting to write all global variables starting with the first one will only serve to write the first byte of the first variable and a portion of the stack "above" the variables in memory (probably program code).

The second complication involves reading the variables back: if the initial write to disk wrote more than required, when the data is read back the extra data will overwrite areas of storage which were not intended to be modified. This could have confusing side effects on some of the other program variables.

The easiest way to solve both of these problems is to create a single record to contain our global variables, and write it to disk using the BLOCKWRITE intrinsic. Since each block is 512 bytes long, we would like our record to be a multiple of 512 bytes long. We can check to see how long it is (using the SIZEOF intrinsic) to make sure that it is an exact multiple of 512. We can then write "SIZEOF(global_rec) div 512" blocks to disk.

Writing the Heap to Disk

The Heap is a thornier problem. Where is it? How big is it? Luckily UCSD includes two intrinsic functions to help us. The Heap is controlled via a "Heap Pointer" which contains the address of the next byte which can be allocated if a request is made for dynamic memory via the NEW intrinsic. The MARK intrinsic allows us to determine the current Heap pointer value, and the RESTORE intrinsic allows us to set that pointer. If we do a MARK during program initialization we will know where the Heap pointer was before the program started gathering "state"; if we MARK just prior to the save, we will know the final Heap pointer value. Note that these pointer values may be either byte or word addresses depending on the version of the operating system; for example, version I.5 uses byte addresses on the Z80 while version III.0 uses word addresses on the MicroEngine. Some implementation dependent code will be required here, but this is taken into account in the sample program which follows.

Simple arithmetic can determine the length of the Heap used, so long as the length is less than 32767: lengths greater than that will be computed correctly in an unsigned arithmetic sense, but they will appear to be negative to Pascal. Attempts to compute the number of blocks to transmit will be incorrect in sign (negative) and in magnitude.

We now have all that we need to do a save or restore of a program's state. The following program demonstrates that the INIT, SAVE, and RESTORE routines function properly. To use it, compile it and do a SAVE with it; run the compiler or some such (to mess up dynamic memory); then run it again and do a RESTORE with it; the same program state should be recovered without error. As you can see, the program was written to give error messages wherever appropriate. This technique should work on all UCSD systems including Versions I.3, I.4, I.5, II.0, II.1, and III.0; I believe that it will work for Version IV.0, but I am not certain of this.

Note: UCSD Pascal is a trademark of the Regents of the University of California.

```pascal
program state_save_restore;

   type
      { dummy dynamic variable to show technique works }
      a_type = array [1..300] of integer;
      mark_type = ^integer;

   var
      { must be a multiple of 512 bytes long to prevent interference with
        run-time stack when read back in during restore }
      globals : record
         pointer_type : (byte_address, word_address);
         old_mark : mark_type;
         new_mark : mark_type;
         root : ^a_type;
         trash : array [1..252] of integer;
         end;
      { the following is not part of the state }
      ch : char;

   procedure write_mark( title : string; some_mark : mark_type);
      var
         { allows us to "see" the value of a pointer as an integer }
         trix : record case boolean of
            true : (i : integer);
            false: (p : ^integer);
            end;
      begin
      trix.p := some_mark;
      writeln(title, trix.i);
      end; { write_mark }

   procedure init;
      var
         r, s : record case boolean of
            true : (p : ^integer);
            false: (i : integer);
            end;
      begin
      writeln('Size of GLOBALS in bytes is ', sizeof(globals));
      new(r.p); { clear off old directory if there }
      { test to see how far heap pointer moves when integer is allocated }
      mark(r.p); new(s.p); mark(s.p);
      if (abs(r.i - s.i) = 1) then begin
         globals.pointer_type := word_address;
         writeln('Word Address Machine.');
         end
      else if (abs(r.i - s.i) = 2) then begin
         globals.pointer_type := byte_address;
         writeln('Byte Address Machine.');
         end
      else begin
         globals.pointer_type := byte_address;
         writeln('WARNING: Unable to determine address mode of machine.');
         end;
      { must be done before program starts using Heap }
      mark(globals.old_mark);
      write_mark('Initial Mark = ', globals.old_mark);
      end; { init }

   procedure save;
      var
         fname : string;
         save_file : file;
         len, i : integer;
      begin
      if (memavail < 600) then
         { must have about 512 bytes slop between Heap and stack in case
           read (during restore) writes on top of dynamic stack area }
         writeln('WARNING: cannot Save safely unless MEMAVAIL > 600.');
      mark(globals.new_mark);
      write_mark('Save Mark = ', globals.new_mark);
      repeat
         write('Enter save file name: '); readln(fname);
         if (length(fname) = 0) then exit(save);
         fname := concat(fname,'.state');
         {$I-} rewrite(save_file, fname); {$I+}
         until (ioresult = 0);
      { write GLOBALS area first }
      if ((sizeof(globals) mod 512) <> 0) then
         writeln('WARNING: SIZEOF(GLOBALS) should be a multiple of 512 ',
            'but it is ', sizeof(globals));
      write('Saving...');
      { note: if GLOBALS is HUGE (> 32767 bytes) then SIZEOF will be negative
        and the following computation will fail miserably ! }
      len := (sizeof(globals) + 511) div 512;
      write(len, ' blocks of globals...');
      i := blockwrite(save_file, globals, len);
      if (i <> len) or (ioresult <> 0) then begin
         writeln; writeln('Error during save.'); end;
      { now write Heap area }
      { note: if the total size of the saved heap is greater than 32767,
        then the following computation will fail miserably. }
      len := ord(globals.new_mark) - ord(globals.old_mark);
      if (globals.pointer_type = byte_address) then len := (len + 511) div 512
         else len := (len + 255) div 256;
      write(len, ' blocks of Heap...');
      i := blockwrite(save_file, globals.old_mark^, len);
      if (i <> len) or (ioresult <> 0) then begin
         writeln; writeln('Error during save.'); end;
      close(save_file,lock);
      writeln('Done.');
      end; { save }
```

Saving Program State under UCSD Pascal continued . . .

```
procedure restore;
   var
      fname : string;
      i, len : integer;
      restore_file : file;
      junk_mark : mark_type;
   begin
   repeat
   write('Enter restore file name: '); readln(fname);
   if (length(fname) = 0) then exit(restore);
   fname := concat(fname, '.state');
   (*$I-*) reset(restore_file, fname); (*$I+*)
   until (ioresult = 0);
   if (i <> len) or (ioresult <> 0) then begin
      writeln; writeln('Error during save.'); end;
   writeln('Done.');
   { re-establish correct value in Heap pointer }
   release(globals.new_mark);
   { test if correct }
   mark(junk_mark);
   write_mark('Restored Mark = ', junk_mark);
   end; { restore }

procedure set_array;
   { put values in array to see if they can be read again after restore }
   var
      i : integer;
   begin
   new(globals.root);
   for i := 1 to 300 do globals.root↑[i] := i;
   end; { set_array }

procedure write_array;
   { display array for user to see }
   var
      i : integer;
   begin
   writeln;
   writeln('Contents of Dynamic Array Follow:');
   for i := 1 to 300 do begin
      write(globals.root↑[i]:4);
      if ((i mod 20) = 0) then writeln;
      end;
   writeln;
   end; { write_array }

begin
init;
repeat
   write('Restoring...');
   { save old initial Heap pointer value in "junk_mark" }
   junk_mark := globals.old_mark;
   { read globals area first }
   { see note in save procedure }
   len := (sizeof(globals) + 511) div 512;
   write(len, ' blocks of globals...');
   i := blockread(restore_file, globals, len);
   if (i <> len) or (ioresult <> 0) then begin
      writeln; writeln('Error during restore.'); end;
   { check to see that memory addresses match between current run
     and data as saved on disk previously }
   if (junk_mark <> globals.old_mark) then
      writeln('WARNING: old and new Heap pointers do not match.');
   { now read Heap area }
   { see notes in save procedure }
   len := ord(globals.new_mark) - ord(globals.old_mark);
   if (globals.pointer_type = byte_address) then len := (len + 511) div 512
      else len := (len + 255) div 256;
   write(len, ' blocks of Heap...');
   i := blockread(restore_file, globals.old_mark↑, len);
```

LOOK NO FURTHER.................................. WE'VE GOT YOUR PRICE!

ALTOS COMPUTERS	RETAIL	PREPAY
5000-15 Single User (64K, 2-1MB 5" disks)	$2995	$2399
5000-5D Single User (64K, 1MB + 5MB 5" disk)	$5995	$4799
8600-10 8-User (512K, .5MB + 10MB disks)	$10500	$8399
AVL 'EAGLE' SERIES DESKTOP COMPUTERS		
Eagle II Desktop System (64K, dual 400K disk)		$2995
Eagle III Desktop System (64K, dual 800K disk)		$3995
Eagle IV Desktop System (64K, 800K + 7 .5MB disk)		$5995
for 'Acctg. Plus' & 'Spellbinder' W/P ADD		$2000

COLUMBIA DATA PRODUCTS COMPUTERS	PREPAY
NEW! Multi Personal Computer (128K, 2-400K disks, 8-slots)	
(IBM PC LOOK-ALIKE - far greater value)	$2995
1503-2 System (128K, dual 800K disks)	$2845
1802-1 System (128K, dual 1MB 8" disks)	CALL!
CONCEPT 1000 - *MultiBus* - up to 16 users	CALL!
CORVUS DESKTOP COMPUTERS	
32-bit Mainframe power in a desktop system..	
NEW! 'Concept' (256K, 68000, 120X56 CRT) from	$4995
Disk subsystems (6, 11 & 20MB) from	$2695

DIGITAL EQUIPMENT CORP. (DEC)	PREPAY
Destined to become legends in their own time..	
'Rainbow 100' Z80/8088 w/CP/M-80/86, 132X24CRT	$3495
Professional 300 series (F11 CPU) from	$3995

DYNABYTE S100 COMPUTERS	RETAIL	PREPAY
NEW! 'Monarch' 6600 (256K, 10MB Disk, etc.)	$10995	$9995
NEW! 'Monarch' 6900	$11995	$10895

MORROW 'DECISION' S100 COMPUTERS	RETAIL	PREPAY
Micro Decision (64K, dual 200K disks, Word Stat, etc.)		$1495
'Desktop' model (65K, 2-400K 5" disks)	$4195	$3495
'Rackmount' model (65K, 2-500K 8" disks)	$4375	$3695
Discus M10 (10MB 5" rigid disk subsystem)	$2795	$2495
Discus M16 (16MB, 5" rigid disk subsystem)	$3495	$2995
Discus M20 (20MB 8" rigid disk subsystem)	$4795	$3995
A822S Dual Slimline 8" floppy pkg. (2MB)	$1995	$1695

N.E.C. DESKTOP COMPUTERS	RETAIL	PREPAY
Hold on to your hats . . . the 'Advanced Personal Computer' has arrived! **Desktop computing will never be the same! (IBM DATA MASTER LOOK ALIKE)**		
NEW! 'HO1' 128K, 8086, 1MB 8" disk GREEN screen		$3295
NEW! 'HO3' 128K, 8086, 2MB 8" disks COLOR		$4995
Options: 128K RAM **$600**		
Graphics from $600, 8087, 2S + 2P I/O		$250
PC8001 Pers. Computer or PC8031 Dual disk	$995	$700
PC8012 I° & RAM Expansion Unit	$649	$500
NON-LINEAR SYSTEMS PORTABLE COMPUTERS		
KAYPRO II a better value than either **Osborne or Otrona!** includes: *9" 80x24 CRT*, two *195K* disks, 64K, Z80, CP/M, 'Select' W/P, ProfitPlan, SBASIC, etc.		$1995
KPRO 5 (same with 5MB rigid disk)		$4495

SANYO DESKTOP COMPUTERS	RETAIL	PREPAY
MBC-1000 'Creative Computer' 66K, Z80, 300K disk	$1995	$1595
BDF-160 Single 300K Add-on Drive Pkg.	$995	$895
BDF-360 Dual 300K Add-on Drive Pkg.	$1595	$1339
BDF-860 Dual 1MB 8" Add-on Drive Pkg.	$2795	$2395
MBC-2000 64K Desktop System (2-300K disk)	$3495	$2995
MBC-3000 64K Desktop System (2-1MB disk)	$4995	CALL!

SYSTEMS GROUP S100 COMPUTERS	RETAIL	PREPAY
System 2814 (64K RAM, 2MB Disk)	$5890	$4599
System 2824 Two User (128K RAM, 2MB Disk)	$6980	$5449
System 2819 (64K RAM, 11MB Disk)	$8905	$6949

TELEVIDEO COMPUTER SYSTEMS	RETAIL	PREPAY
TS-800 64K Satellite Wrkstn. for TS-806	$1795	$1399
TS-802 Desktop System (64K, 2-800K disks)	$3495	$2699
TS-802H Similar with 7.5MB Rigid Disk	$6995	$5299
TS-806 Computer (64K, 800K + 7.5MB disk)	$6995	$5499
'TeleSolutions' (WordStar & CalcStar)	$495	$400

VECTOR GRAPHIC S100 SYSTEMS

Vector offers ON-SITE service & the most powerful W/P & spreadsheet software available on CP/M systems. TOP RATED in user surveys for over three years!

System 2800 - dual 1MB 8" drives	IN STOCK!	$5495
Vector 4/20 (128K, 2-600K disks, Z80B + 8088)		$4495
Vector 4/30 (128K, 600K + 5MB disks, Z80B + 8088)		$5995

With Memorite III Included

If you can find a better value than Vector - BUY IT!

XEROX DESKTOP COMPUTER SYSTEMS	RETAIL	PREPAY
'SAM' 820-1 (64K, dual 92K drives)	$2995	$2395
'SAM' 820-2 (64K, dual 8" 256K drives)	$3795	$2995
'SAM' 820-II (64K, dual 8" 1MB drives)		CALL!
Add **$169** for CP/M		

VIDEO DISPLAY TERMINALS	RETAIL	PREPAY
ADDS 'Viewpoint' series from..	$695	$589
ADDS 'Viewpoint 60' emulating terminal	$895	$799
AMPEX Dialog-80 **CLOSEOUT SPECIAL**	$1195	$699
DEC VT-101	$1650	$1099
DEC VT-131	$2450	$1669
DataGraphix 132-1 Hacker Specials from..	$2000	$299
SCION MicroAngelo Color & B/W Terminals..		$2249
TeleVideo 910	$699	$589
TeleVideo 925C	$995	$799
TeleVideo 950C	$1195	$949
NEW! TeleVideo 970		CALL!
NEW! TeleVideo 'Intelligent I' (w/64K RAM)	**$1695**	$1399
Wyse WY-100 -or- ALtos 'SmartTerm'	**$995**	$899
Wyse WY-200 (132x25)		$1199
Zenith Z19 CRT Terminal (VT52 compatible)	$895	$799

MATRIX PRINTERS	RETAIL	PREPAY
NEW! Anadex DP9501A Printer (200 cps)	$1725	$1399
NEW! Anadex DP9620A (300/150 cps) DP/WP Printer		$1599
DEC Letterprinter 100 (30/80/240 cps) DP/WP Prtr.		$2699
Florida Data OSP120 (600/150 cps) Printer	$3900	$3399
Flor. Data OSP120 (600/150 cps) printer	$4200	$3699
ID'S 'Prism' 4-color printer (150/200 cps)	$1995	$1699
IDS 560-G Printer	$1395	$1199
Tally MT160-I series (40/160 cps) from..	$845	CALL!
Tally MT1800 series (50/200 cps) Printers	$1995	$1639
Okidata 82A (120 cps - RS232 & Cent.)	$649	$499
Okidata 83A (120 cps - wide carrage)	$995	$799
Okidata 84 (200 cps - Dot Plot Graphics)	$1495	$1199

LETTER QUALITY PRINTERS	RETAIL	PREPAY
Diablo 620 Printer (25 cpa)	$2000	$1699
Diablo 630 Printer (40 cps)	$2900	$2099
Diablo 630 KSR Terminal (40 cps)	$3325	$2749
NEC 3510 (RS232(or 3530 (Cent.) Printer	$2295	$1899
NEC 7700 Printer (O.E.M. parallel)	$2995	$2399
NEC 7710 (RS232) or 7730 (Cent.) Printer	$3095	$2499
NEC 7720 KSR Terminal (55 cps)	$3610	$2899
Qume Sprint 9/35 (35 cps) Printer	$2160	$1899
Qume Sprint 9/45 (45 cps) Ltd. Panel	$2495	$2099
Qume Sprint II		CALL!
TEC (C. Itoh) F10 Printer (40 cps)	$1995	$1499
Single Bin Sheet Feed or Envelope Feed Mechanism	$1550	$1100
Dual Bin Sheet Feeder Mechanism	$2300	$1600
Envelope Feeder Mechanism	$1495	$1100
(Please specify NEC, Qume, Diablo or TEC model)		
CLOSEOUT - Qume Sprint 3 & 5 'SpeedFeed'	$1295	**$599**

SOFTWARE 'BEST-SELLER' LIST	RETAIL	PREPAY
Ashton-Tate 'dBASE II' Relational DBMS	$700	$530
'dBASE II' for 8088/8086		CALL!
Aspen 'ProofReader & Grammatik'	$279	$229
BPI Acctg., Job Cost or Inventory Modules	$395	$319
CIS COBOL Compiler	$850	$750
Digital Research PL/1-80	$500	$425
Digital Research 'Pascal MT+'	$475	$425
DMA 'THE FORMULA' Applications Generator	$595	$500
Ecosoft 'MicroStat' Statistics Library	$295	$269
Fox & Geller 'Quickscreen' Screen Bldr.	$149	$135
InterSystems 'PASCAL-ZB' Z80 PASCAL	$395	$350
Lexisoft 'Spellbinder' Word Processing	$495	$300
MarketFax Sales & Marketing Tools	$500	$425
mDBS III with DRS, QRS & RTL Utilities		$3000
MicroPro WordStar w/ MailMerge *& Tutorial*	$665	$429
MicroPro 'DataStar' File Manager	$350	$235
Microsoft BASIC-80 Compiler	$395	$325
Microsoft FORTRAN-80 Compiler	$500	$400
Microsoft COBOL-80 Compiler	$750	$600
NECHE 'Select' or NECIS 'Benchmark' W/P	$495	$400
NECHE 'Report Manager' Electronic Wksht.	$200	$169
NWA 'Statpak' Statistics Library	$495	$400
NPI Law Office Mgr. w/ G/L & Docket Control		$4000
Oasis 'The World Plus' Speller (50K words)	$150	$109
Open Systems Acctg. & Inv. (per module)	$995	$795
Palantir Word Processing Package	$425	$350
Peachtree 'PeachPak 4' (GL, AR, AP, PR, INV)	$2475	$2000
'Reformatter' CP/M to IBM -or- DEC formats		$200
'Select' w/'SuperCalc' W/P Package	$595	$400
Sorcim 'SuperCalc' Electronic Worksheet	$295	$219
Sorcim 'Pascal/M' for 8086 and 8088	$495	$400
Sorcim 'TRANS-86' Translator for 8086	$125	$109
Sorcim 'ACT-86/88' (8086/8088 X-assembler)	$175	$139
Star Systems Acctg. (GL, AR, OE, AP, PO, PR)	$1600	$1100
Star Systems Property Management	$950	$799
Systems Plus 'Acctg. Plus' (8 modules)	$4760	$3500
Systems Plus (Any One Module)	$595	$469
Target PlannerCalc gridsheet		$50
TLB Solomon I Acctg. System (MDBS based)	$2595	$2200
TLB Solomon Series II Job Cost Acctg.	$3495	$3000
Vector 'VPS' Series (MEM3, EPL2, etc.)	$1585	$2350
Vector 'CONECT' Communications Software		$150
VectorVector CP/M2.5 128K Upgrade (w/64K RAM card)		$895
(much more available - please call)		

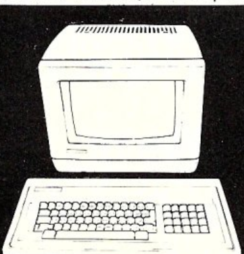
S100 & MultiBus Circuit Cards - 10% to 20% off

IMB offers the FINEST SELECTION and **BEST VALUES** in computers, peripherals and software, for the discriminating buyer. Our affiliate, Budget InfoSystems, is an 'Authorized Distributor' for many product lines and offers 'Full Service' for each - **COMPLETE WARRANTY PROTECTION IS ALWAYS ASSURED!**

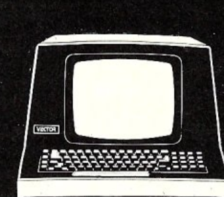
IBM specializes in service to **FOREIGN, PROFESSIONAL, MILITARY, GOVERNMENTAL, INSTITUTIONAL, and O.E.M.** markets! Call for details regarding quantity discounts and low cost leasing plans.

IBM values all of its customers, but we must encourage the untutored novice to purchase from a local '*Authorized Dealer.*' to preserve product integrity. Feel free to contact our office if no local dealer is available. We must insist that 'Turn-key Systems' be purchased through our 'Full Service' associate. **Budget InfoSystems**, to assure proper customer support.

TERMS: PREPAYMENT   **CONTACT**: Scott Secor, Gen'l Mgr.
F.O.B.: MPLS MN   **Delivery**: Stock to 4 wks. A.R.O.



International Microcomputer Brokers
607 NE Highway Ten—Jefferson Center
(across from NorthTown Mall)
Blaine, MN 55434
(612) 780-5361 -or- 786-5545
Hours: Daily 10AM to 6PM CST



A Comparison Between Stoic and Forth

by Richard H. Mossip

Stoic is a development from Forth. It uses a very similar structure, and much of the syntax is identical. Stoic in its present form can be best regarded as an enhanced form of Forth specifically adapted to CP/M. In brief, Stoic is faster and easier to use, while Forth is somewhat smaller, and available on many different CPUs in a standardised form.

History

Stoic was written as a combined project at Harvard Medical School and MIT for biomedical applications. A version of this with some adaptations to CP/M appears in the CP/M users group library. It has a file system, but it is not compatible with CP/M. The version of Stoic discussed here, Stoic-XB, was made fully compatible with CP/M and considerably extended by Jeff Zurkow of Avocet Systems. I have made further modifications to improve user convenience and add some extra features.

File System

Forth does not have a file system as such. Disk space is addressed as sequential 1K blocks. Stoic-XB is file-oriented, and is totally CP/M compatible. The file system in Stoic-XB operates through CP/M and allows for creating, opening, closing, deleting and getting the size and directory files. There are also sequential and random access commands on a byte-by-byte basis both for high-level code and as machine code callable subroutines. File operations can be either buffered or unbuffered. Any number of file control blocks can be created, and simultaneously open.

Forth source code is written in 1K "Screens"—each is a 16 × 64 video screen image. This results in larger source code than a file-oriented system, as TABs and short lines do not save space. This results in a tendency to skimp on comments so as to pack the maximum code onto a screen. As a result this produces code which is hard to understand later on, and to maintain. Stoic-XB source files are similar to assembly-language source files in that there is little penalty for "prettyprinting" to show loops and conditionals clearly, and include extensive comments.

Editor Convenience

Both languages have an editor built in as part of the language. The Forth editor is usually a very simple screen editor. The Stoic-XB editor is a very powerful general purpose text editor modelled after the NOVA editor, with command changes to be compatible with CP/M's ED. It is much more convenient to use. Backup files are automatically maintained by the editor in Stoic-XB, and not at all in Forth.

Number of Commands

Forth is smaller than Stoic partly because there are fewer commands. For example, Forth has 5 comparisons, Stoic 16. These take up space, but make application programs shorter. The commands are listed by section to give a clearer picture.

Speed of Execution

75% of the definitions in Stoic are in directly executable machine code. The other 25% are in high-level code. Forth is almost exactly the other way round, with 25% in code and 75% in high-level. As a result of this, Stoic programs execute faster, as they are usually closer to machine code. On the Interface Age prime number benchmark the times were (2MHz Z80):

FORTH	303 seconds
STOIC-XB	179 seconds (41% faster)

This comparison was made executing identical code on the same machine.

Dictionary Structure

Forth can have up to 31 significant characters in a name. Stoic truncates at 5 and keeps the length. This can lead to ambiguity if one is careless (input1 and input2 are the same, 1input and 2input are not.) The Forth dictionary is a single linked list. The Stoic-XB dictionary is broken into sections with a hash algorithm which makes loading (compilation) 8 times faster for Stoic-XB. This is important when cross-compiling long application programs.

Execution from the Keyboard

Stoic has a compile buffer into which typed-in definitions are compiled before execution. As a result

conditional and loop commands can be executed immediately. Forth lacks this feature, and so conditionals and loops can only exist in colon definitions. This makes testing definitions from the keyboard a great deal easier with Stoic, as there are no restrictions.

Stacks

Forth has two stacks. The parameter stack stores data, and the return stack stores addresses for nested high-level definitions. Loop limits are also stored on the return stack, making the loop index only available at the same level of subroutine nesting as the loop itself. This makes it easier to mess up the return stack inadvertently and get lost. Stoic maintains an entirely separate loop stack for keeping loop indices and other purposes, making it less likely to become contaminated.

Assembly Language

Both Forth and Stoic have basically similar assemblers. Most implementations of Forth have an 8080 assembler, but Z80 versions are also available. The Stoic-XB assembler has several which mimic the high-level control structures of IF ... ELSE ... END IF BEGIN ... END BEGIN .. IF .. REPEAT. This makes it very easy to code an already written high-level routine to make it run faster. In addition most of the math operations exist in Stoic-XB as callable subroutines as well as high-level commands. Some disk access subroutines are also provided.

Strings

These are a standard data type with Stoic which makes string handling easier, and also the format for definitions more consistent (the name can precede the colon in true RPN fashion).

Double-Precision Integer and Floating Point

These data types are available as standard modules for Stoic. They are also available for some Forth implementations.

Transportability

Forth is implemented in a standardised form on many processors, including most Micros and many Minis. Stoic is at present only available on 8080/Z80/8085 systems. This is important if you use several CPUs, not if you don't. It is easier to write Forth for a new CPU as there is less machine code involved.

Conclusion

Stoic and Forth are very close. The differences are ones of degree not kind. It would be possible to write all the Stoic features into Forth. You would then have Stoic! Similarly you could remove features from Stoic and get Forth.

In a system which uses CP/M and file operations I feel there are clear advantages to Stoic-XB. If you need to have programs directly translatable between CPUs and file operations are not important, Forth would be quicker.

It is much more convenient to write programs in Stoic, due to the file structure and better editor. I also feel that for utility programs which stand alone the Stoic-XB cross-compiler is easier to use than the Forth one I have examined. This may well be influenced by the fact that I am familiar with it and know it works well. The Forth one I have looked at, but not yet used in earnest.

The Stoic Language

Why Stoic was selected as the development language for the control program of a new scientific instrument

by Richard H. Mossip

Introduction

About 18 months ago I started on the design of a new scientific instrument intended as a smart replacement for a stripchart recorder. Stripchart recorders move a pen across a sheet of paper in response to a voltage signal, while the paper is moved past the pen at constant speed to produce a graphical record. They are used widely in laboratories for making continuous measurements during an experiment, often for long periods while they are unattended. We felt that with the availability of high-quality, low-priced dot-matrix printers such as the Epson MX80 (which has excellent positional accuracy for graphics), a stripchart recorder could be made using a microprocessor to provide many features which would be very useful, and had never been available before.

The Hardware

The instrument is designed around an 8085 microprocessor. The 8085 was chosen because it is well suited to a moderate-sized application with a lot of EPROM program memory and I/O but only a small amount (4K) of RAM, so the Z80's ability to simply refresh dynamic RAM was not needed. There was also available a commercial single-board system (the Explorer, from Netronics Inc., New Milford, CT) which had the right mix of memory and I/O, as well as an S-100 bus interface which would allow system expansion in the future. This choice also allowed us the unusual luxury of running a CP/M system on the target hardware during development, for program debugging. This proved to be extremely useful, and saved us much time.

To minimize development time and cost, we decided to use commercially available assemblies wherever possible. As mentioned above, a single-board CPU system was selected (after examining about 50 possibilities) and an S-100 EPROM card was used for program memory. The multiplexed 16-bit Analog-Digital converter and the keyboard/display modules were specially developed, as no suitable units existed.

Richard H. Mossip, 13 W. Buena Vista Way, Bloomingdale, NJ 07403.

The Software

The novel features of the instrument lie in the software, which is quite extensive (26K of object code for the basic instrument). The user can print up to 12 traces as a stripchart recorder, with the digital values for all 12 and the time (either real or elapsed) printed alongside. The printer is interrupt-driven, using one of the hardware vectored interupts on the 8085; the optional RS-232 interface uses another. The real-time clock is maintained with the non-maskable interrupt.

The program is divided into two parts, setup and runtime. The setup portion interacts with the user to get the gain and sensitivity for each measurement channel, whether a channel is to be printed as an analog trace or a digital printout, or both, the speed of the chart etc. This is done with very extensive prompting and printed messages in case the instruction manual is lost (or not read!). There is also a fast method for frequent users. An editing mode is provided for changing a program which is already set up.

The runtime portion controls the analog-digital conversion, averages the measured data to reduce noise, and formats it into a buffer which is sent to the printer as a string of graphics characters. It also monitors the keyboard while the chart is printing and provides commands for changing any parameter while the chart is running. The new value is printed on the chart as a record of the change and when it was made.

Optional features include a fast RAM data buffer to capture events occurring faster than the printer can record them, and also a built-in user-available language emulating a programmable calculator with a program which is run each time a measurement is taken. This allows complex data manipulation in between measurement and printing (we even have a program for demonstrations which prints pretty charts without making measurements . . .). Other features available are an RS-232 interface and cassette program storage.

Selecting a Language

One of the most important tasks was to select the language to be used in developing the software.

Speed of development was more important than minimizing object code size, as EPROM costs were expected to come down, and very high production volume is not anticipated. This indicates the desirability of a high-level language. Some routines clearly would have to be in assembly language (the actual measurement routines, interrupt handlers, etc.) so ease of interfacing to machine language modules was important. Because the program was to be put in EPROM, runtime overhead should be minimized.

Development was started using Microsoft Basic, with the intention of compiling when it was running to increase speed and reduce the object code size. However, when it was discovered that the runtime overhead for a program containing nothing other than an end-of-file marker was in excess of 8K, this clearly was not the optimum choice. With one-third of the program written, the compiled object code occupied 26K, and the source was too long for the interpreter in a 48K system. At this point I began a search for a better way. The following table shows various parameters important to this application (my wish list) rated on a scale of 0 (worst) to 5 (best) for several languages. These ratings are a personal opinion, and yours may well differ!

Language	High Level	Fast	Compact	Assembler Interface	Ease of use
Basic	5	0	0	2	5
Compiled Basic	5	3	2	3	1
Pascal	5	3	3	2	2
Forth	5	3	5	5	4
Stoic	5	4	4	5	5
PL/I	5	3	2	3	1
Macro-Assembler	1	5	5	5	0

The language chosen is not available directly to the user, as the system is menu-driven, so general understanding of the language was unimportant.

Software development is much more rapid with an interpreter than a compiler. One can debug the code and make immediate corrections, without having to use a separate editor, compiler, linking loader, and debugger, all of which take several minutes to load and run, and the whole process has to be repeated for each correction.

A compiler should be able to produce much more compact code than an interpreter, and also execute faster, so reducing the amount of machine code routines which had to be written. Intermediate code languages such as CBASIC and UCSD Pascal seemed to me to combine the development disadvantages of a compiler with a large runtime overhead, and so were rejected.

At the other end of the scale, assembly language is fast and compact, but the development time for a complex program such as this was expected to be too long. This time could be reduced by macro languages such as ML80, but not to a great extent, as the macros would have to be written also.

The best choice seemed to lie between Pascal (with a 'naked code' compiler), Forth, Stoic and PL/M. C was also a possibility, but at that time a suitable compiler was not available. It also tends to be clumsy for floating-point math.

Forth and Stoic both have the great advantage of being interpretive in nature, which makes for fast debugging, yet both produce fast, compact code. Either of these was felt to be far preferable for this project to a compile-only language. Cross-compilers which could produce stripped-down object code which could be placed in EPROM were available for both languages.

We decided on Stoic for several reasons, and felt Forth was the next best alternative. Stoic is an outgrowth of Forth, written at MIT and Harvard Medical School. It has the same structure, and much of the syntax is similar. The internal structure is slightly different from Forth. Stoic has 4 stacks, not 2, which makes it more resistant to crashing by programming errors. More instructions are built into the language. The principal difference is that Stoic is a file-oriented language, while Forth accesses the disk as 1K blocks in a virtual-memory format. This may seem minor, but it has an important impact on programming style. With Forth, a line always occupies 64 characters, and a 16-line page 1K whether you use it all or not. This means that there is a tendency to compress a lot of code onto one page at the expense of clear structured layout and full comments. This frequently results in programs which are difficult to understand later, and are time-consuming to maintain. Program maintenance often is a significant cost over the life cycle of a product.

With Stoic, as with a regular assembler, only the characters used appear in the file, and there is less pressure to save space. Program layout is usually indented to show the structure, and comments can be added freely. A very convenient and powerful text editor, modelled on the NOVA, is part of the Stoic language. It is already in RAM, so you can enter or leave the editor in fraction of a

Forth and Stoic are both interpretive in nature, yet both produce fast, compact code.

second. As with a Basic interpreter, you can test a portion of a program, correct an error, then try it again in a few seconds. You can also try out a routine from the keyboard in 'immediate' mode, one piece at a time, to locate problems, and also inspect or change any variable.

Because Forth and Stoic are 'Stack machines,' subroutines normally communicate with each other using the stack. They thus require far fewer variables than other languages. In PL/I and Pascal, each procedure has its own set of local variables. With Stoic most of these can be eliminated, and thus save considerable amounts of RAM.

A second major reason that Stoic was selected over Forth was the cross-compiler or target compiler. This compiler takes the program, after testing on a development system, and produces compact ROMable code which can be built into the final instrument. The Stoic cross-compiler was designed so that the same program source files could be used, either in the resident (development) system or cross-compiled simply by changing the short loader file which loads each of the source code modules. This minimizes problems which show up on cross-compiling. It was felt that this transition could be made more easily with the Stoic cross-compiler rather than with Forth. The Stoic system used (Avocet System, 804 S. State St, Dover, DE 19901; $2,000) is totally CP/M-compatible both for loading and for its file structure. This fitted in well with our development system, and was compatible with existing software and utilities. A third reason was the availability of a fast floating-point math package.

Getting Started

I started out with Stoic using the version in the CP/M Users' Group library (Volume 23). This has some disk I/O grafted onto the MIT program. It works fine, but the file and disk structure is incompatible with CP/M. The documentation is quite extensive, and reasonably clear in the applications areas of the language. The inner workings are sketchily explained at best. This is only a problem, however, if one wishes to make major modifications to the language itself, which is something one cannot do at all in most languages. The area which is missing (and even more missing for Forth) is a tutorial publication indicating how one should write programs effectively in the language. Program structure is quite different from that of Basic and many other languages, as the use of subroutines is much more extensive.

One thing which seemed a little strange at first was the Reverse Polish Notation. I had strongly preferred the 'bracketed' format as I was used to it. However it only took a week or so to be completely converted to what I now feel is a much easier and more logically structured notation. There are two plusses for Reverse Polish Notation (RPN). One which is unarguable is the much faster execution. When a language such as Basic scans an expression to see how it should be evaluated, whenever it finds a bracket it must halt, look for further brackets before the matching closing bracket, and evaluate from the innermost pair of brackets outwards. This is clearly much more complex and thus slower than the RPN approach of carrying out each command as it is encountered. In RPN, the order of executing commands (such as +,−,*,/ etc.) is set when the expression is written, and there is no ambiguity. This also eliminates the commonest cause of syntax errors with Basic (missing parentheses) as they are not needed at all! The other plus is that once one has gotten used to the different notation, it does seem much clearer, and it is easier to figure out what is happening with a complex expression.

The use of a stack is also an important feature, which ties in closely with RPN. This behaves just like a machine language stack, onto which you can push data, and then pop it off at a later time. In Stoic (and Forth) each operation pops its input data off the stack, and pushes the result on the top of the stack. For example, the operator + removes the top two items from the stack, adds them together, and returns the result to the top of the stack. All the other arithmetic operators act similarly. As a simple comparison, here is the same expression with parentheses and in RPN:

(A + B) * (C + D)	A B + C D + *
11 operators	7 operators

A, B, C, D are numbers (literals). In Stoic, entering a literal causes its value to be put on the stack, so when the + operator is encountered the stack contains A and B. These are added together,

Intermediate results can be "buried" on the stack for later recovery. This use of the stack almost completely eliminates the need for local variables.

and the stack is left with the sum A+B. C and D are also literals, so when the second + is reached the stack contains three items: A+B, C and D. The + takes off C and D and returns C+D. The multiply operator, *, takes off two numbers, A+B and C+D, and multiplies them leaving the desired result on the stack for the next operation. This illustrates an important feature of stacks: intermediate results can be 'buried' on the stack and recovered when needed later. To do this there are several specialized stack manipulation commands to rearrange the stack. This use of the stack almost completely eliminates the need for local variables.

Another area in which Stoic differs from many other languages is that it gives you complete control over the machine. You can perform any operation on either addresses or on data, and use absolute, relative or labelled addresses. You have complete power, but with power also comes responsibility. It is up to the user to ensure that he is operating on the correct data. Because data is put on the stack by earlier operations, Stoic does no type checking. If you add something to the address of a variable instead of to its contents, you will merely get the wrong answer. If you store data at the value of a variable instead of at its address, then by Murphy's law the data will probably go into the middle of the language and crash the system, requiring you to reload it from disk. The usual symptom of this is Stoic complaining that some perfectly reasonable command is 'undefined', because the thread linking the dictionary became broken. Checks for such things could have been incorporated but would have slowed the language down and made it less flexible. If you want more checks, you can always redefine the language, using the language itself. . . .

This may seem somewhat like Alice in Wonderland but is really quite simple. All the commands (your program and the language also) are linked together as a list. When scanning a command this list is searched starting with the most recent definition. The first match that is found is used. If you give a new function the same name as an existing one, Stoic will tell you you are redefining it as a warning. The new name is used for all later definitions, while the earlier one is still used for everything up to the redefinition. I used this feature for the first programming I did in Stoic to change the editor commands so as to be compatible with those for CP/M ED.

What Stoic Programs Look Like

Programs written in Stoic have no line numbers. All references are by name. The language stores the length and the first 5 letters only of the name in its dictionary, in order to conserve space. If reasonably descriptive names are chosen, the program becomes almost self-documenting in a similar way to CBASIC. Structured programming is strongly supported, and Stoic does not even possess a "GOTO" command. When you design a program in Stoic, you start from the top by deciding what the major modules are to do, and how they relate to each other. For example, in our application, the word which calls everything else is a continuous loop, as we never want to give the user access to Stoic. The major modules are:

INITIALIZE sets up variables, tests memory, etc.
SETUP gets the gain and zero for each printed channel from the user
MEASURE measures all the active channels
TIMEOUT checks to see if we should print a row of data
SCALE scales the data for printing
PRINTOUT formats the data for the printer
STOP? checks the keyboard for the END key

```
"Dianagraph"
: INITIALIZE BEGIN
     SETUP BEGIN
       MEASURE TIMEOUT
         IF SCALE PRINTOUT ENDIF
     STOP? END
   0 END ;
```

A definition in Stoic starts with a colon and ends with a semicolon. Each word in the definition is executed, in turn, as a subroutine, and must previously have been defined. We first define each word as a dummy stub that just returns the results expected by the other routines, and then flesh them out in detail as the design progressses. BEGIN . . . END is a loop structure that repeats until the word before END leaves a non-zero number on the stack, when the loop exits. The outer loop will continue forever because a zero is put

You can define your own data structures to perform a particular task . . . Another feature of Stoic is the case of linking to machine code.

on the stack just before END. This is the last definition in the program and to execute it from the keyboard one types DIANAGRAPH. In the ROM version, it is executed automatically at power-on. This form of organization means that the interface between modules is tested automatically as program development continues, and one does not have to fit everything together at the end of the project. The overhead for a subroutine is very small, so to minimize space a large number of short subroutines are used. When cross-compiled, the overhead for each subroutine is 4 bytes plus 2 bytes for each call. Thus, a sequence of more than 4 instructions used twice, or 2 instructions used 4 times, uses less space in subroutine form. If speed is important, the tradeoff changes, but the time overhead is also quite short as the overhead for each high-level definition is about 50 microseconds.

New Data Structures

You can also define your own data structures to perform a particular task. We had to print out a great many different instructions to the user during the course of setup, as we wanted the instrument to be usable even when the manual was lost. To do this without excessive memory usage, we built a dictionary of the words to be used, saving further space by limiting the number of words in the dictionary to 256 so that a byte could be used as a pointer. This was not a serious limitation however, as messages with many words that are used only once can be put in the regular string format, and the wording can often be slightly changed to eliminate rarely used words. Each word thus took one byte as a position in the table, rather than a byte for each character.

We found it easy to make the building of the dictionary automatic during program loading. A word "[" was defined to indicate the start of a message and "]" to indicate the end. As each word in the message is encountered it is looked up in the dictionary and, if found, the word number is returned. If the word was not found, it is added to the end of the dictionary. The message is followed by a command to print it (TMSG). A typical source code entry for a message is:

[PRESS ENTER WHEN READY] TMSG

A useful utility was added to print out an alphabetical listing of the dictionary, together with the number of uses for each word, as an aid when dictionary pruning was needed.

Program Control

The control constructs used by Stoic are IF.. ELSE..ENDIF; BEGIN...END; DO...LOOP (similar to FOR..NEXT in Basic). All of these folow the structured programming philosophy that a loop should have only one entry and one exit. This takes a little getting used to, but makes for programs which are quite readily understood, particularly when combined with indented layout to show the nesting of loops and conditionals. The programs are built up of 'words' which you define and are added to the language, and are in essence subroutines. These subroutines can be nested to any desired depth because of the stack structure of Stoic.

Mixing High-Level with Assembler

Another feature of Stoic is the ease of linking to machine code. If you want to write a routine in machine code, the instruction CODE< turns on the built-in assembler. At the end of the routine you push onto the stack any data to be returned, and jump to NEXT to return to the Stoic runtime interpreter. This routine is treated by Stoic identically to any other routine. It is transparent to any routine using it whether the routine is in machine language or in high-level. This is a degree of flexibility which is hard to beat.

When the program had been designed and tested, it was found that certain parts of it (which were concerned with scaling and formatting the data to be printed) did not run fast enough. The inner loops of these portions were rewritten using the Stoic assembler. This was particularly easy since the assembler has built into it (essentially as macros) control constructs which mimic those of the high-level language. There is an assembler IF...ELSE...ENDIF, a BEGIN...END (both of which can test for any flag). This provides structured forward jumps within the same code definition (a word defined in assembly language). A nice feature of Stoic is that all the CPU registers are available to you without restriction within a code definition. Most implementations of Forth have one register committed to a pointer to the next in-

Without Stoic it would have been difficult to achieve as much in the time available.

struction which you must preserve, while in Stoic this pointer is maintained as a variable in memory.

Writing a Language with Stoic

Another part of the software for this instrument involves providing user-programmable math operations on the data between measurement and printing, for applications such as integration, differentiation, averaging, and linearization, and also to provide flexible alarm and control functions. It was decided to provide this as a language, and to emulate a programmable calculator so that the user need not know a computer language. Stoic is well adapted to this, since the user program is entered into an area of memory as a list of routine addresses in their execution order. This list is then executed by the normal Stoic runtime interpreter.

To build the list, the keyboard entries are compared with a command table, which returns the command number if a match is found. The runtime address is found from an address list maintained in the same order. This approach was found to be simpler than the usual Stoic linked vocabulary, because all instructions have 3 letters, and in the cross-compiled version the regular vocabulary operations are absent. The "stack" for the RPN calculator is maintained as 4 variables, to prevent the possibility of an incorrect user program crashing the system stack by leaving things on it or taking too much off. Program editing features and a full trace mode, with complete printout of the stack at every operation, were added as debugging aids. The high speed of the Stoic runtime interpreter is important in this application, as the program is run every time a row of dots is to be printed on the chart (1/2 second minimum at present, but we expect to speed it up to 1/8 second). This system has been tested in the field against dataloggers operating under Basic and, as expected, operated much faster for real-time data analysis.

Conclusion

We have completed development of the basic instrument, and have shipped the first units. We are currently working on some of the options, and in retrospect feel that Stoic was the correct language choice for our project, and that it would have been difficult, if not impossible, to achieve as much in the time any other way.

Stoic is ideal for medium or large programs. If it is important that the size be under 3K or so, the overhead would be too heavy, and assembler should be used. With the cross-compiler one can generate stand-alone utilities very easily, and the result is almost disassembler-proof because fragments of machine code are mixed in with address lists, and all the names are removed!

Having used Stoic for over a year, I feel that it is the ultimate hackers' language. With Stoic, you can combine the best features both of Basic for quick turnaround and of machine language for speed. Anyone who is familiar with programming in assembler will find it very easy to use, and the high-level portions are as easy to use as BASIC. The only snag for a novice is that, apart from checking for empty or full stacks and for gross syntax errors such as failing to terminate definitions, Stoic takes few steps to protect you from yourself. The editor is so good that I use it for everything apart from word processing. It is more powerful than any other I have seen (including Z-TEL !) because all the features of Stoic are available to you within the editor, including taking commands from multiple disk files.

Software Directory

If you have a software package that you are offering for sale and would like to get some free publicity, send us a description (about 120 words) that follows the format of this column. We reserve the right to edit or reject any submission.

Also, we do publish reviews of CP/M based software products. If you would like to have your product reviewed, then you must furnish a review copy for this. Please call us first, before sending the review copy to us.

Program Name: Improved Fixed Point Package for Pascal/Z.
Hardware System: Any system capable of running Ithaca Intersystems Pascal/Z.
Minimum Memory: Same as Pascal/Z.
Language: Z80 assembler. To be used with Pascal/Z.
Description: Contains replacements for all routines in original Intersystems fixed point package, written intersystems fixed point package, written mostly in assembly language. All routines are ROMable and re-entrant. Routines are up to 60 times faster and take up only about 2.2K bytes when assembled. (Only 1.5K if Realtofix and Fixtoreal are removed.) They can be set for 2 to 105 digits of precision. All known bugs in the intersystems version have been corrected.
Release: January 1982
Price: $50.00: Documentation only $2.00 (refundable with purchase).
Included with Price: Source file with and without comments, documentation, test/demo program.
Where to Purchase It: Brom Microsystems Engineering, Inc., Box 616, Winona, MN 55987-0616. (507)-452-5805

Program Name: REVAS
Hardware System: Z80 CP/M system
Minimum Memory Size: 32K
Language: Machine Code
Description: This powerful disassembler produces .ASM and .PRN type files and listings for reassembly, editing, or analysis. Supports ZILOG and TDL mnemonics, and the undocumented Z80 opcodes. Disassembles memory resident code, or program files as long as 64K bytes.

Permits command abbreviations, command strings, command macros, and on-line help. Commands use symbolic arguments, or numbers in radix. I/O files can be interactively allocated; output goes to your choice of console, files, and printer.

Full symbol table editing provides complete control over disassembly. Symbols and or data types are assigned automatically or interactively. Analytical aids include insertion, calculation, search and cross-reference functions, and status displays of user and REVAS assigned parameters.
When released: January 1982
Price: $90.00
What is included with price: Manual, SSSD 8" Disk or SSSD NorthStar or Micropolis Mod II
Where to purchase it: Revasco, 6032 Chariton Ave., Los Angeles, CA 90056. (213) 649-3575

Program Name: Renumber for North Star Basic
Hardware System: 8080/Z80 North Star Basic
Minimum Memory Size: 32K

Language: Object code
Description: "Renumber" expands North Star's Basic Renumbering command so that the programmer can: (1) Renumber and/or move selected lines of a basic program. (2) Open up program and add more code than the correct line numbering sequence will allow. (3) Keep specific line numbering sequences and renumber the rest.
When released: February, 1982
Price: $39.50 + $2.00 shipping
What is included with price: Diskette and documentation
Where to purchase it: E.T. Software Services, 1072 Casitas Pass Rd, Carpinteria, CA 93013. (805) 684-8259

Program Name: Scan for North Star Basic
Hardware System: 8080/Z80 North Star Basic
Minimum Memory Size: 32K
Language: Object code
Description: Adds a new command NorthStar Basic called "Scan." With the scan command the user has the ability to scan a basic program and selectively list only those program lines that contain a variable, word, group of words, line number reference or anything that is contained in any basic program line.
When released: February, 1982
Price: $29.50 + $2.00 shipping
What is included with price: Diskette and documentation
Where to purchase it: E.T. Software Services, 1072 Casitas Pass Rd, Carpinteria, CA 93013. (805) 684-8259

Program Name: macroP
Hardware System: 8080/8085/Z80 CP/M and 8086 CP/M-86
Minimum Memory Size: 56K
Language: Object Code
Description: macroP allows you to add new, more powerful commands to your document formatter or language processor. For the assembly language programmer, macroP converts simple assemblers into macro assemblers. For the high-level language programmer, macroP adds indefinitely long variable names, compile-time expression evaluation, conditional compilation, and time and date stamping of source programs. For the text formatter user, macroP adds conditional nested inclusion of text files and automatic numbering of section headings. macroP is a general-purpose macroprocessor, styled after those available until now only on mainframe and minicomputer systems.
When released: May 1982
Price: $135.00, postpaid within the U.S.
What is included with price: One 8″ SD diskette, manual, and sample macro files for programmers and users of Textwriter.
Where to purchase it: Pluto Research Group, P.O. Box 50444, Palo Alto, CA 94303-0444. (415) 323-5654

Program Name: Cobol Compiler (ANSI 74)
Intel 8086/8088 hard disk + 8″ floppy
Minimum Memory Size: 96K
Language: 8086 machine language
Description: Cobol compiler ANSI 74 Standard (low intermediate).

Produces real executable 8086 machine code which is runtime and memory efficient. Detailed error handling (over 500 messages). Requires Digital Research CP/M-86 operating system.
When released: May 1982
Price: $1400

What is included with price: Manual
Where to purchase it: mbp Software and Systems Technology, Inc., 7700 Edgewater Drive, Ste 626, Oakland, CA 94621. (415) 632-1555

Program Name: dBgen
Hardware System: CP/M, Z80, 8080 system
Minimum Memory Size: 56K
Language: machine code
Description: dBgen translates screen and menu layouts directly into dBASE II code so users can go from concept to running program in minutes. A formatted database (.DBF file) is created automatically and a report describing its structure printed. Variables are defined by enclosing their names in braces where they are to appear on the screen when the program is executed. dBgen produces a set of DO files that can be modified, included in other programs or run unchanged. The menu drive program can also be used to format reports.
When released: April 1982
Price: $95.00
What is included with price: Program with Documentation
Where to purchase it: Active Computer Enterprises, 1953 E. Apache Blvd., Tempe, AR 85281. (602) 968-3350

Program Name: BASIC/Z Compiler
Any 8080/Z80 CP/M (rev. #2.x) system
Minimum Memory Size: 48K recommended
Language: Object Code
Description: BASIC/Z is an interactive compiler. It supports cursor addressing (CRT and printer), reverse video, blinking fields, erase to end-line and end-screen, clear screen, and more. Screen-oriented editing at runtime includes non-destructive cursor movement, character deletion, insert/change modes, and dynamic character limitation. Edit control codes are user-definable.

All floating-point operations are performed in decimal (BCD), avoiding conversion errors common to binary systems. Additionally, the accuracy of computation and storage may be program-defined to any level, with up to 18 digits of precision. Arrays may be dimensioned dynamically (by an expression), and even erased to reclaim memory space.

In addition to alphanumeric

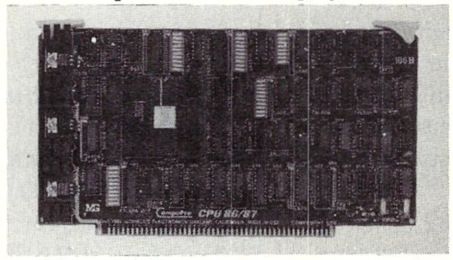
labels, BASIC/Z supports recursive, multiline user-defined functions. The self-contained SORT verb will sort 2000 element in 2 seconds. Multitiered error-trapping handles BDOS errors. It includes a full function program editor, which tests syntax as you type! An extensive debugging facility provides line trace, error line retention, and the unique ability to "single-step" a compiled program, with continuous display of selected variables.
When released: 1982
Price: Single-site use license, $345.00
What is included: BASIC/Z compiler, RUN/Z runtime module, TR/III translator utility, INSTALL menu-driven installation package, documentation.
Where to purchase it: System/z, Inc., P.O. Box 11, Richton Park, IL 60471. (312) 481-8085

Program Name: BACKUP
Hardware System: 8080, 8085, or Z80 CPU running CP/M
Minimum Memory Size: 48K
Language: N/A
Description: BACKUP is a powerful utility program that enables the user to back up his/her hard disk utilizing inexpensive floppy disks. The ability to back up files that are larger than the capacity of the floppy disk makes BACKUP unique to the CP/M 2.x user. Included with BACKUP is a poweful File Directory program and a utility to test your existing system to see if BACKUP will operate properly.
When released: April 1, 1982
Price: $95.00
What is included with price: 8″ Floppy disk and manual
Where to purchase it: TRI-L DATA SYSTEMS, 900 Fort Street, Suite 50 Honolulu, Hawaii 96813. (808) 524-3780

New Products

8086/87 S-100 CPU Board

An 8086/8087 microprocessor board providing 16-bit capability with provisions for adding a mathematics co-processor and operating system firmware has been introduced by CompuPro.



Compatible with IEEE 696/S-100 standards, CPU 86/87 is available in either 8- or 10-MHz microprocessor versions. Accommodating 8- or 16-bit words, its on-board logic can read or write two bytes serially for 8-bit applications, or pass word-wide values for 16-bit operation. Users can mix 8- and 16-bit devices in the same system.

The board accepts Intel's 8087 math processor and 80130 operating system firmware ICs. The 8087 offers a high-speed number crunching capability, while the firmware adds an 8-level vectored interrupt controller, three interval timers, and a choice of silicon-based operating systems: the iRMX-86 kernel or CP/M-86.

The 86/87 CPU generates a full 24-bit address for its 16-Mbyte memory, and a power-on-jump capability allows jumping to any 4k boundary in the lower 1-Mbyte address space. A clock-switching circuit permits slave processors to share a bus with the board, thereby eliminating bus conflicts by running the slave and the master at different clock rates.

The CPU 86/87 comes configured with microprocessor, a ROM-less version of the 80130, and a socket for the 8087 math processor. Suggested retail price is $695 for the 8-MHz and $850 for the 10-MHz version. CompuPro, Oakland Airport, CA 94614; 415-562-0638.

S-100 256K-Kilobyte RAM CARD

Dual System's new DMEM/256KP, Dynamic Memory Board provides 256 kilobytes of memory on a single IEEE Standard 696/S-100 Bus Compatible Board. Thus it makes available four times as much memory in the same area as previously available 64K-byte boards.

The memory board can be addressed to provide either 8- or 16-bit memory transfers. Left and right 8-bit transfers can be juxtaposed using an onboard jumper. Access time is 230 nanoseconds and cycle time is 580 nanoseconds, including transparent refresh.

The board is organized in

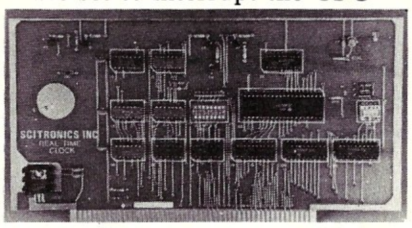
two independent 128-kilobyte

regions; thus each region can be viewed as an independently-addressed portion of memory. Addressing of up to 16 Mbytes of memory is provided via extended, 24-bit addressing. Address selection is by means of DIP switches which establish boundaries on each 128K region in the address space.

Parity is checked on every byte transfer. If any parity check is found to be invalid, the board supplies an error signal in the form of a pulse or latched bus error. Two LED indicators are mounted on the board to indicate transfer activity and a latched parity error. Dual Systems Control Corporation, 720 Channing Way, Berkeley, California 94710, 415/549-3854.

S-100 Time/Calendar Board

A newly-developed Real Time Clock provides an S-100 computer with complete timing and calendar information including seconds, minutes, hours (12 or 24 hour format), year, month, date and day of week data. The clock can be read upon command or can be software/hardware set to interrupt the CPU



every hour, minute, second or 1/1024th second. It utilizes a crystal-controlled LSI CMOS clock chip for high accuracy. An on-board, lithium battery provides power back-up for up to 6000 hours of computer power failure. BASIC and assembly language programs to set and read the clock, are provided. Price: $179.00.

1200-bps Modem

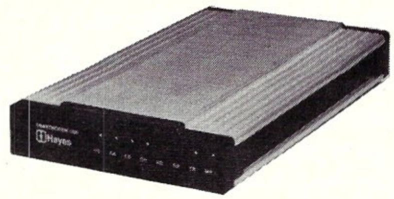
The Hayes Stack® Smartmodem 1200 is a Bell 212A/103 compatible modem that lets RS-232C compatible computers or terminals communicate over telephone lines at 1200 bps or 0-300 bps.

The Smartmodem 1200 connects directly to the telephone line and an RS-232C port, and is approved by the FCC for direct connection to any U.S. telephone system for both pulse and Touch-Tone* dialing. Both

types of dialing may be combined in a single command with pulse used, for example, to access a PBX board and Touch-Tone used to dial an outside number.

The Smartmodem 1200 is an intelligent system that executes user commands and responds with either decimal digit or English word result codes. The modem can be controlled by any programming language and it includes all circuitry for auto-dial and auto-answer.

Users hear the progress of a call via an audio monitor and are alerted to wrong numbers and busy signals. A repeat command causes automatic redialing of a number. Indicator lights on the modem's front panel allow a visual check of its operational status. Switch and program-controlled options include full or half duplex, enable auto-answer, and result code type. Commands allow selection and change of additional parameters such as dialing speed, escape code character, and number of rings before the modem answers a call.



Price: $699; Hayes Microcomputer Products, Inc., 5835 Peachtree Corners East, Norcross, GA 30092, (404) 449-8791.

Microsystems Mart

Please Direct Inquiries to: Jim Beloff; (212) 725-4216.

LEO ELECTRONICS, INC.
8921 S. Sepulveda #208
Los Angeles, CA 90045
(213) 641-3101 (800) 421-2418

WHY PAY MORE? SAVE MONEY!
LOWEST PRICES ON PARTS!

EPROMS	
2708	3.00
2716	4.00
2716-1(350ns)	6.50
TMS 2716	6.00
2732	7.50
2532	8.00

REGULATORS	
7805	.75
7812	.75
7815	.75
7905	.85
7912	.85
7915	.85

RAMS	
4116(200ns)	8/13.00
4116(150ns)	8/16.00
2114(200ns)	8/18.00
4164(200ns)	8/54.00
4164(150ns)	8/56.00

MICROPROCESSORS	
Z-80A-CPU	5.50
8080A	3.00
8085A	7.00

TERMS: Check, Visa, Mastercard. Call for C.O.D. U.S. Funds only. California Residents add 6.5% Tax.
SHIPPING: Add $2.00 for Ground $5.00 for Air.
ALL MAJOR MANUFACTURERS
ALL PARTS 100% GUARANTEED

NORTH STAR Advantage Users:
NORTH STAR Horizon Users:

New software for Advantage CPM Users KEY MASTER for NORTH STAR Computers — Program any key on your Advantage keyboard! Ideal for word processors or spread sheets. Use the function and cursor—movement keys. KEY MASTER is menu-driven and user-friendly. The Horizon version will be available soon. Introductory price KEY MASTER: $49.95.

For NORTH STAR DOS 5.2 or GDOS USERS: Your system may be used as a super intelligent terminal! Communications programs running under DOS 5.2DQ or GDOS allow for serial communication via the RS232 port. Incoming Data may be saved. Programs may be downloaded to the basic interpreter. Text files can be edited, printed & exchanged with other computers. Source listings available as an option. INTRODUCTORY PRICE: $29.95 (For either version).

MASTERCARD/VISA ACCEPTED
To obtain information or to order:
THE COMPUTER SHOP
283 Medford Ave.,
Patchogue, N.Y. 11772
Telephone: (516) 758-6558

LOOK TO THE FUTURE...

EPROM-32

The only EPROM programmer you need!

- IEEE-696 (S-100) EPROM programmer for single supply (+5V) EPROMs.
- Programs current 1K through 16K (byte) EPROMs plus future 32K EPROMs.
- EPROM can be verified through a port or located in memory space for verification and use.
- Personality Modules adapt board to different EPROM types:
  PM-1—2508, 2758, 2516, 2716 PM-2—2532 PM-3—2732, 2732A PM-4—2564 PM-5—2764 PM-6—68764 PM-7—2528 PM-8—27128
- Single zero-insertion-pressure socket accommodates both 24-pin and 28-pin EPROM packages.
- On-board DC-to-DC converter with adjustable regulator for programming voltage.
- Double-sided PC board with solder masks, silkscreen and gold-plated contact fingers.
- 8080/Z80 control software includes commands for programming, verification, disk I/O and editing.
- Comprehensive user's manual contains source listing of control software.

MicroDynamics Corporation
P.O. Box 17577
Memphis, TN 38117
(901)-755-0619

$269.95
(assembled & tested)

Price includes EPROM-32, documentation and two Personality Modules (specify). Additional Modules—$7.95. Control software on 8 inch single density CP/M-compatible diskette—$9.95.

MasterCard & Visa

TN residents add 6% sales tax.

WORD PROCESSING SOFTWARE
$45.

A unique modular set of programs designed to make your word processing chores faster and easier.

Programs Include:
*Text formatter printing utility
*File Transliteration
*Word and Character Counting
*File Encryption/Decryption, and more
*9 Programs in All.

For CP/M 2.2 8 inch single density disk systems, 48Kram or more. Users manual only $3.00 (Applicable towards purchase).

Send Check or Money Order to:
L.A. SOFTWARE
6708 Melrose
Los Angeles, CA. 90038
California Residents Add Sales Tax.

SAVE 90%

YES you can save up to 90% on a computer system of your own.

$150.00 buys a 4MHz Z80A with 64KB & a real Front Panel

$200.00 buys a Full Function 24x80 CRT with Keyboard

You can have your own computer and be running Fortran, Basic, Pascal, etc. If you get our

FREE BROCHURE
TODAY

DIGATEK CORPORATION
Suite 40
2723 West Butler Drive
Phoenix AZ 85021

ROLL-YOUR-OWN TECHNOLOGY AND SAVE A BUNDLE

64K Dynamic RAM ICs

Quantity	200 ns 64K Ram	150 ns 64K Ram
0 - 49	@ $6.99	@ $7.99
50 - 149	@ $6.75	@ $7.75
150 +	@ $6.50	@ $7.50

HM6116P-3 150 ns 2KX8 static @ 5.50
2732 450 ns 4KX8 EPROM @ 5.00
IN STOCK: PRIME
OKI MSM3764-20RS
FUJITSU MB 8264-15
OR CAN GET ANY
JAPANESE BRAND OR FLAVOR

Send Cash, M.O., COD or VISA, M/C To:
MICRO PROCESSORS UNLIMITED
24,000 SOUTH PEORIA AVENUE
BEGGS, OKLAHOMA 74421
918-267-4796

IMPORTANT: Call for current prices, other brands, other speeds, or volume pricing. I WANT your business, call any hour (almost). Prices shown include cash discount. Shipping & insurance extra. These prices are subject to change!

"DISC DOUBLER PROGRAM"

"DOUBLE THE STORAGE CAPACITY ON YOUR SS/SD OR SS/DD 5¼ INCH DISKETTES."

NOW YOU CAN INCREASE THE SIZE OF USEABLE SPACE ON YOUR SS/SD OR SS/DD 5¼ INCH DISKETTES.

ALL YOU PAY IS A ONE TIME FEE OF $12.95. USE THIS PROGRAM OVER AND OVER, AS OFTEN AS YOU LIKE, NO EXTRA FEES. THIS PROGRAM WILL WORK WITH ALMOST ALL SS/SD OR SS/DD DISKETTES. HURRY, SEND $12.95 U.S., CASH, CHECK, OR MONEY ORDER. VISA AND MASTERCARD BUYERS TELEPHONE TOLL FREE.

DISK DOUBLER PROGRAM
CP-M COMPANY
P.O. Box 1045 DEPT. CCO
WOODLAND HILLS, CALIFORNIA 91365
Nationwide 800-428-7825 Ext. 43
CA. Residents Only 800-428-7824 Ext. 43

California residents add 6% sales tax, Los Angeles residents add 6.5% sales tax. Please include $1 postage and handling. All orders shipped within 10 days. Disk doubler program copyright 1982 by computer peripheral-material company.

NOW AVAILABLE, CRT STAND
FOR XEROX 820-822-860 COMPUTERS

FINALLY, A GOOD LOOKING QUALITY STAND ENABLING XEROX 820-822-860 USERS TO ELEVATE AND PIVOT THE CRT WHILE IMPROVING AIR CIRCULATION.
UNIQUE DELTA DESIGN, GIVES MAXIMUM SUPPORT USING LESS DESK SPACE THAN THE CRT ITSELF. STRONG ENOUGH TO SUPPORT 8" DISC DRIVES AND CRT, PIVOTS 180°.

THIS STAND MAY BE USED WITH *ANY* CRT DEVICE AND WILL ELEVATE THE CRT 3⅜", OPTIONAL ADJUSTABLE "FEET" ALLOWS INCREASED HEIGHT IF REQUIRED.

STEEL W/WOOD GRAIN VINYL FINISH $33.95
XEROX "PUTTY" COLOR FINISH $35.95
ALUMINUM W/XEROX "PUTTY" COLOR FINISH $42.95

SEND CASH, CHECK OR MONEY ORDER. VISA AND MASTERCARD BUYERS TELEPHONE TOLL FREE.

COMPUTER PERIPHERALS—
MATERIALS COMPANY
P.O. BOX 1045 DEPT MSO
WOODLAND HILLS, CALIFORNIA 91365
NATIONWIDE 800-428-7825 EXT. 43
CA. RESIDENTS ONLY
800-428-7824 EXT. 43

DEALERS AND DISTRIBUTOR
INQUIRIES WELCOME

California Residents add 6% sales tax. Los Angeles residents add 6.5% sales tax. Can be shipped UPS, PARCEL POST. Weight 3 lbs. In the Continental U.S. add $2.50 postage and handling, $5 elsewhere. Xerox 820-822-860 Trademark and product of Xerox Corp. Patent pending CP-M Company 1982.

N E W

COMPUTER MART
FORMAT
3" x 2"

PICTURE YOUR AD HERE!

NEXT CLOSING DATE:
SEPTEMBER 20

ADVERTISERS INDEX

TEAM PLAYERS.

Master/Slaves...

At Advanced Micro Digital, we believe in co-operation between computers and their operators. That's why our new team of S-100 master and slave processors are designed with your team in mind and feature superlative reliability and ease of operation.

Our team captain is SUPER QUAD, a unique multi-function master processor combining 64K of bank selectable RAM, single- and double-density floppy disk controller, system monitor EPROM, Z80A CPU, two serial and two parallel communication ports on a single board.

The players consist of one or more SUPER-SLAVES, the latest addition to Advanced Micro Digital's line of superior multi-function S-100 boards.

Each SUPER-SLAVE is a powerful single-board slave processor designed for use with the SUPER QUAD in either network or stand-alone configurations. The outstanding features of the SUPER-SLAVE include:

- A DEDICATED Z80A CPU FOR EACH USER
- IEEE-696 standard conformity
- 4 serial, 2 parallel interface ports
- 2/4K EPROM (monitor)
- 64/128K bank switchable RAM
- One year warranty
- Turbo-DOS™, the state-of-the-art operating system with an advanced failure detection and recovery facility that makes the master-slave network virtually crash-proof.
- Also, for the first time CP/NOS® operating system from Digital Research. Plug as many SUPER-SLAVES into the BUS as you need users.

Your team needs the strong support of the SUPER QUAD/SUPER SLAVE team from Advanced Micro Digital Corporation. The Super System includes SUPER QUAD and CP/M operating system. The shugart SA-1000 or quantum Q2000 hard disks are also supported.

ADVANCED MICRO DIGITAL CORPORATION

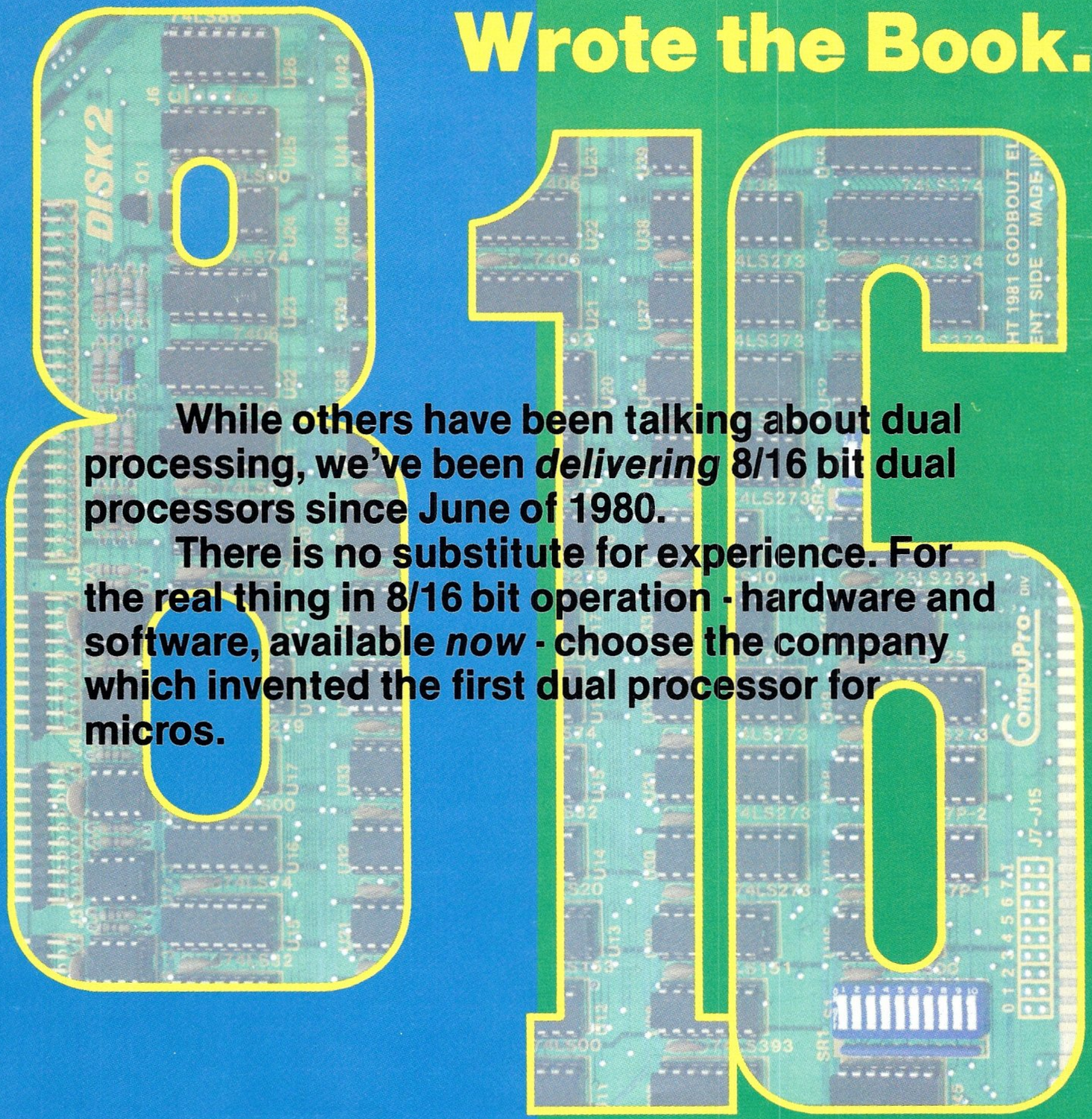
For more information write or call: Sales Dept.

12700-B Knott Street • Garden Grove, California 92641 • (714) 891-4004 TELEX 678401 tab irin

® Registered Trademark of Digital Research Corp
™ Registered Trademark of Software 2000 Inc.

CIRCLE 23 ON READER SERVICE CARD

When it Comes to Dual Processing, We Wrote the Book.



While others have been talking about dual processing, we've been *delivering* 8/16 bit dual processors since June of 1980.

There is no substitute for experience. For the real thing in 8/16 bit operation - hardware and software, available *now* - choose the company which invented the first dual processor for micros.

CompuPro SYSTEMS

CompuPro division, Godbout Electronics, Oakland Airport, CA 94614-0355 (415) 562-0636

CIRCLE 16 ON READER SERVICE CARD